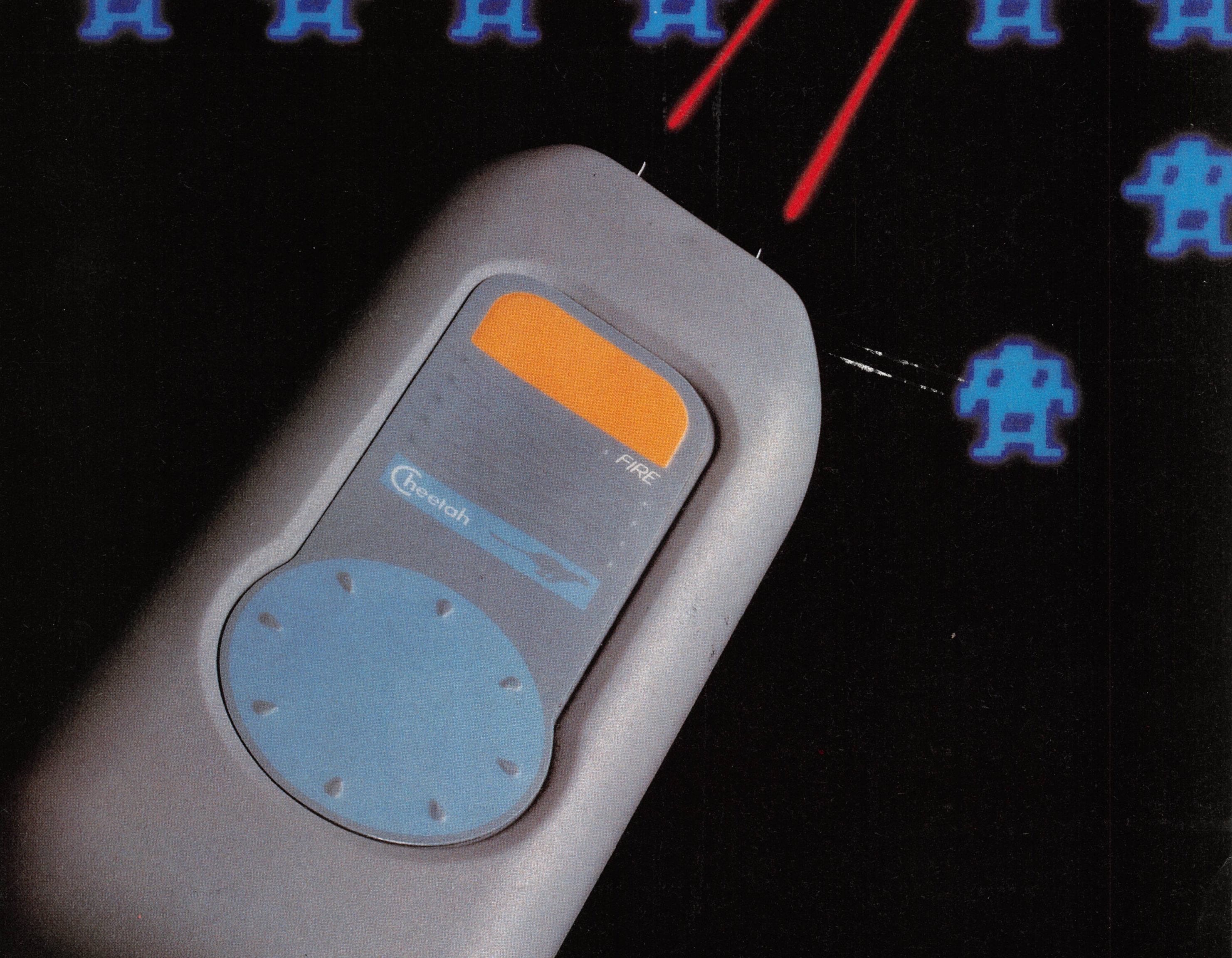
ISSN 0265-2919

80p 30

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- Continuing our occasional series on portable computers, we review Epson's PX-8, the super lap-held with CP/M, telecommunications and RAM disk
- We begin a large-scale survey of micros in movement — robotics at work and play
- Our workshop feature introduces reversed polarity switching.

QUIZ

1) What is dynamic dimensioning?
3) What does 'NOWRAP' do?
3) How is the USR function important in defining graphic characters?
4) What is feedback control?

Answers To Last Week's Quiz

1) A LOGO procedure is said to be state transparent if, after execution, the turtle's position and heading remain unchanged.
2) PRODOS, the new Apple operating system, is unusual in having a hierarchical directory structure, which is accessed via a 'pathname'.
3) MIDI is a standard for digital communications. MIDI compatible devices such as computers and synthesisers can communicate without interfaces.
4) Memotech's first product was a 16-Kbyte expansion board for the ZX81.

COVER PHOTOGRAPHY BY PAUL CHAVE

Editor Jim Lennox; **Managing Editor** Mike Wesley; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Art Editor** Claudia Zeff; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Sub Editor** Steve Mann; **Researcher** Melanie Davis; **Contributors** Geoff Bains, Harvey Mellor, Mike Curtis, Steve Colwill, Rory Forsyth, Graham Storrs; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editors** Chris Cooper, Maurice Geller; **Production Controller** Peter Taylor-Medhurst; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95
How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.
Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited. 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.
How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.
Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

GRAND FINALE

In this final part of the series, we look at some of the advanced systems that have appeared since electronic music began to incorporate digital technology. Many of these systems cost thousands of pounds, but there are signs that, as research costs are recouped and competition increases, drastic price reductions are on the way.

The single most significant recent development in electronic music has been in the area of digital recording. Not only has sound recording quality improved immensely, but the whole meaning of the word 'recording' has changed to embrace a number of different techniques. If we now understand recording to mean 'the digital encoding of sound and its organisation into music', then we can begin to grasp what is happening to music in the 1980s.

Since the Second World War, sound recording on magnetic tape has been the norm, with formats ranging from the tiniest microcassette up to the large spools of 24-track tape used in professional recording studios. When a recording is made on magnetic tape, the minute particles of metal oxide on the tape surface become rearranged in complex patterns that are analogous to the sound waveforms they represent. As the tape passes the playback head of a recording device, these patterns are converted into chains of electrical voltages. These voltages are then fed to the loudspeakers, which reproduce the sound recorded on the tape.

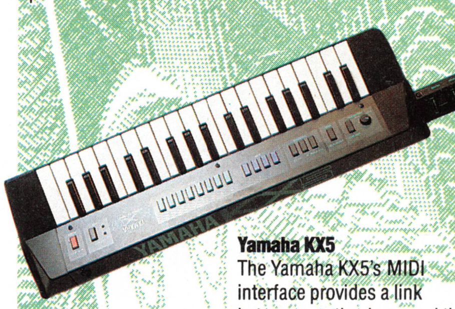
Because the arrangement of particles may be fairly accurately judged in relation to the tape playback head, it is quite easy to find where a particular sound is on a piece of magnetic tape, so the splicing and editing of tape with demagnetised razor blades has become an important skill to be acquired by sound engineers.

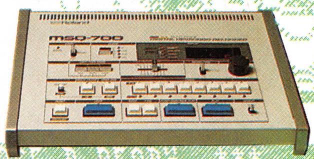
In digital recording, sound is encoded numerically along the tape, and the playback head becomes a digital-to-analogue converter. The loudspeakers are powered in the same way as before, but they use voltages generated by the D/A converter. Provided there is sufficient data for conversion, digital tape can produce reproduction that is greatly superior to that achieved with magnetic tape. And, as long as the data remains intact, the tape may be copied digitally hundreds of times with no loss of quality. With magnetic tape, however, each 'generation' of copying adds hiss and distortion to the recording.

This type of noise degradation has been a familiar problem for sound engineers for many

Synclavier
The Synclavier from New England Digital is considered one of the most advanced in the world. Apart from the usual synthesiser functions, the machine has the capacity to store up to 10 Megabytes of sound on disk

Fairlight CMI
The Fairlight CMI was one of the first computer music systems. The Fairlight's operating system is menu-driven, allowing a variety of options from keyboard control to waveform drawing. The machine also has the facility to produce hard copy printouts


Yamaha KX5
The Yamaha KX5's MIDI interface provides a link between synthesisers and the Yamaha CX5. This device will also be used to interface the Yamaha MSX home computer when it arrives in the United Kingdom


Roland MSQ-700
The Roland MSQ-700 claimed to be the world's first MIDI-compatible sequencer. The MSQ-700 has a full range of MIDI facilities and can store up to 6,500 notes

Drumulator
The Drumulator drum machine from Emu systems has a memory capacity of 10,088 notes within 64 songs. There are also facilities to allow the insertion of additional ROMs that provide additional effects such as Latin or African percussion

HARDWARE COURTESY OF SYCO SYSTEMS

IAN McKINNELL

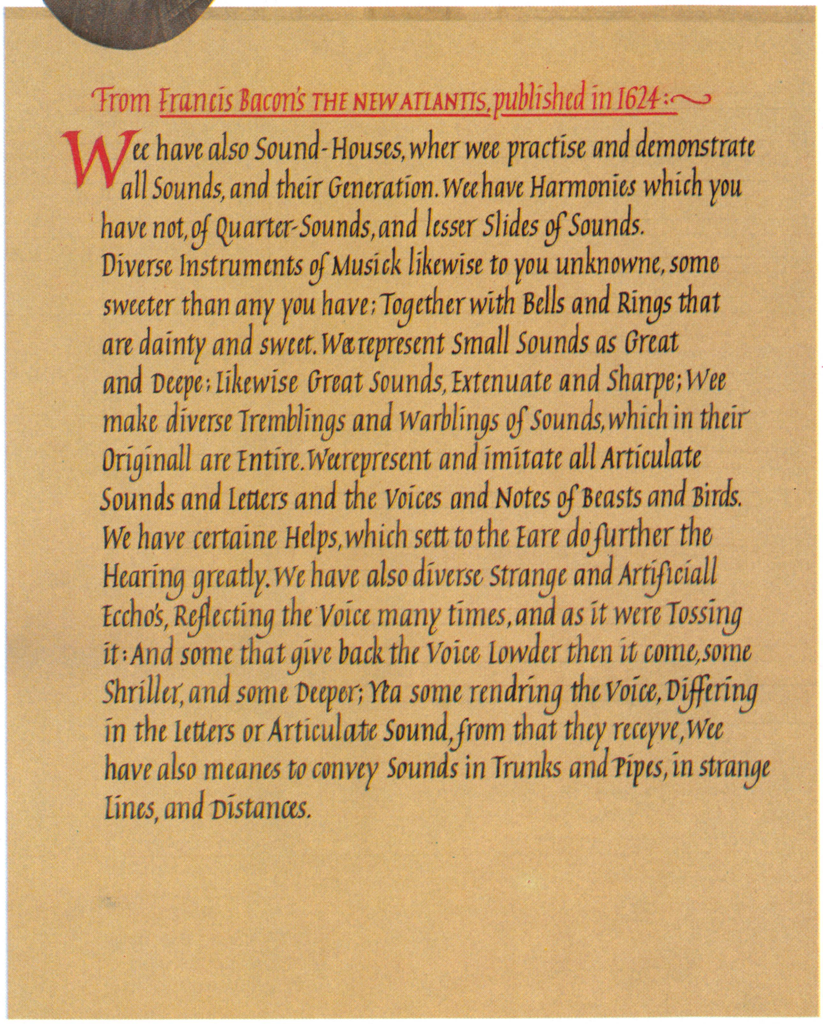
years. Now that this has been overcome, many top recording studios have installed 24-track digital tape recorders. Using these, the sound reproduction is so accurate that engineers find it impossible to ascertain whether a sound coming from the studio monitor speakers is played by a musician in the recording area or is produced by the playback of a digital tape. But new problems have emerged: it is no longer possible to 'see' where sounds are located on a digital tape, which makes editing more difficult, and splicing is becoming an outmoded skill. Another difficulty is 'studio noise', an unwanted and usually unheard characteristic of some audio equipment. Magnetic tape was not sensitive enough to register this, but digital recordings tend to pick it up.

While 24-track recording is still the prerogative of expensive studios, single-track digital recording of the same quality is available to any owner of a Betamax home video recorder. Video tape is a digital medium, and as such it can be used to encode any type of data. The Sony PCM (Pulse Code Modulator) is a unit that converts a Betamax video recorder into an audio tape recorder. This unit, which costs a few hundred pounds only, has the potential to make similarly sized analogue recorders obsolete.

Digital sound encoding, or *sampling,* is at the heart of the Fairlight CMI (Computer Musical Instrument), one of the best known of the advanced systems. The Fairlight can sample any sound for a duration of up to two seconds, and it will then reproduce that sound across a pitch range of six octaves. Sampling is a real breakthrough in electronic music. For years, engineers and musicians have been trying to simulate the sound of strings or woodwinds by using synthesisers, and in some cases they have come very close to reaching their goal. But sampling will provide not only a remarkable reproduction of the sound of 'strings', it can produce the sound of a *particular* violin. Furthermore, in some cases, it can reproduce the sound of a particular player in a particular room. In the first article in this series, we saw how the *musique concrète* composers of the 1950s spent weeks splicing together tiny snippets of recorded tape, eventually producing large-scale pieces. The computer manipulation of samples would now enable a composer to produce similar results in minutes.

Past Echoes
This piece is taken from 'New Atlantis', a vision of utopia written by Francis Bacon (1561-1626). His descriptions of sounds seem to predict the extraordinary power and versatility of today's electronic music

MARY EVANS PICTURE LIBRARY


From Francis Bacon's THE NEW ATLANTIS, published in 1624:~

Wee have also Sound-Houses, wher wee practise and demonstrate all Sounds, and their Generation. Wee have Harmonies which you have not, of Quarter-Sounds, and lesser Slides of Sounds. Diverse Instruments of Musick likewise to you unknowne, some sweeter than any you have; Together with Bells and Rings that are dainty and sweet. Wee represent Small Sounds as Great and Deepe; likewise Great Sounds, Extenuate and Sharpe; Wee make diverse Tremblings and Warblings of Sounds, which in their Originall are Entire. Wee represent and imitate all Articulate Sounds and Letters and the Voices and Notes of Beasts and Birds. We have certaine Helps, which sett to the Eare do further the Hearing greatly. We have also diverse Strange and Artificiall Eccho's, Reflecting the Voice many times, and as it were Tossing it; And some that give back the Voice Lowder then it come, some Shriller, and some Deeper; Yea some rendring the Voice, Differing in the Letters or Articulate Sound, from that they receyve. Wee have also meanes to convey Sounds in Trunks and Pipes, in strange Lines, and Distances.

CALLIGRAPHY BY BERNARD JENNINGS

SAMPLING INSTRUMENTS

A sampling instrument like the Fairlight can overcome the natural limitations of musical instruments. For example, it is quite easy for a flautist to produce a warm, breathy tone quality at the lowest end of the flute's range. However, it is impossible for a player, no matter how skilled, to achieve this type of sound two octaves higher at the top end of the instrument's range — the physical design of the flute prevents this. A Fairlight user, on the other hand, can sample the low, breathy tone and then transpose it upwards by two octaves on the keyboard. The result will still sound like a flute, but it is a type of flute that cannot exist in the 'real' analogue world.

The Fairlight can supply a screen display of any of its sampled sounds, which are stored on 8in disks. Different characteristics of an individual sound may be examined in succession: it is often easier to tell what is 'wrong' with a particular sound by looking at it rather than listening to it. Many sounds need to be longer than their original two-second sample duration. By seeing how the different waveforms inside the sound are related, a point can be selected at which the sound could be made to start *looping,* or repeating. If the right point is selected, this will give the illusion of genuine continuity. Analogue sound doesn't behave like this, of course, so looping can give an unusual dimension to the music being produced.

The Fairlight user has two ways of inputting music, apart from playing in 'real time' at the keyboard. The first method, known as 'Page R', gives a five-line stave display, and the user enters notes onto the stave from the music keyboard. Any timing errors are 'tidied up' automatically by the computer in accordance with the metre or time signature that the user has already specified.

The second method is to use an MCL (music composition language). The Fairlight MCL demands that every note-event should be entered via the alphanumeric keyboard, but it enables time and accentuation to be changed on a note-for-note basis. The Fairlight has an eight-voice capacity, so a user may enter eight different sequences, played by eight different sounds or 'instruments'. Each of these may be slightly out of time — by a matter of milliseconds — with the others, and the whole performance is co-ordinated by the Fairlight's internal clock.

It may well be asked what use it is for music to be performed in this 'inaccurate' way, especially by a computer. The answer is that people never play exactly in time, and one of the elements of performance — especially among jazz and some classical players — is the way in which a musician may 'bend' musical time when executing a passage. An operating system like the Fairlight provides a way in which certain styles of performance can be simulated. These simulations can be used in experimental and research work, just as computer simulations are used in the design of car bodies, aircraft wings and Space Shuttle heat-shields.

Many musicians are justifiably concerned that instruments such as the Fairlight will replace people, especially as the simulating capacity of such equipment develops. Groups like Wang Chung, Duran Duran and Culture Club use Fairlights as part of their production process, and it is often impossible to tell what is actually being played and what is Fairlight-performed. However, once a user gets to know the operating system, it becomes clear that the Fairlight is much more than just an expensive mimicking device. It is a genuinely new musical instrument, and has a potential that is still largely unexplored.

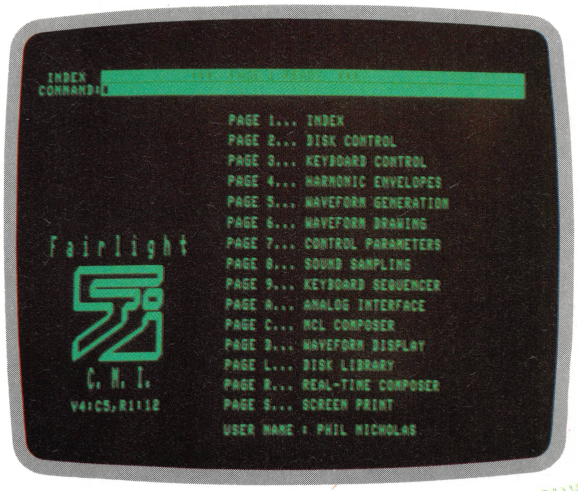
Although it is the best known, the Fairlight is not the only such instrument available. At twice the price of the Fairlight, the Synclavier system (costing around £30,000) has similar facilities, but on a larger scale. Data is stored using hard Winchester disks with a 40 megabyte capacity. An entire album could be produced and recorded within the Synclavier system itself, making an advanced 24-track digital machine completely unnecessary.

But even the Synclavier has its limitations. At present, all the available sampling instruments have one thing in common — they reproduce the sampled sound as close to the original as possible, unless the user has intervened to make a specific modification. But a trumpeter, in the middle of a live performance, can make a considerable difference to the next sound to be played simply by changing breath control or lip position. A competent player does this almost without thinking. The next step for sampling instruments would therefore be to produce a 'player-responsive' system.

The Kurzweil, which is still a prototype, incorporates a pattern-recognition program. This means that when a note is played on the music keyboard a number of different samples are scanned, and characteristics from each sample are combined to produce the individual sound. The type of characteristics selected should reflect the way in which the music is played. Such a system would increasingly resemble the character and feel of a real acoustic instrument. The only difference is that no one upright or grand piano is exactly like another. All Kurzweil CMI 'pianos' will be identical in character and feel — unless users develop a method of writing their own 'character and feel' software.

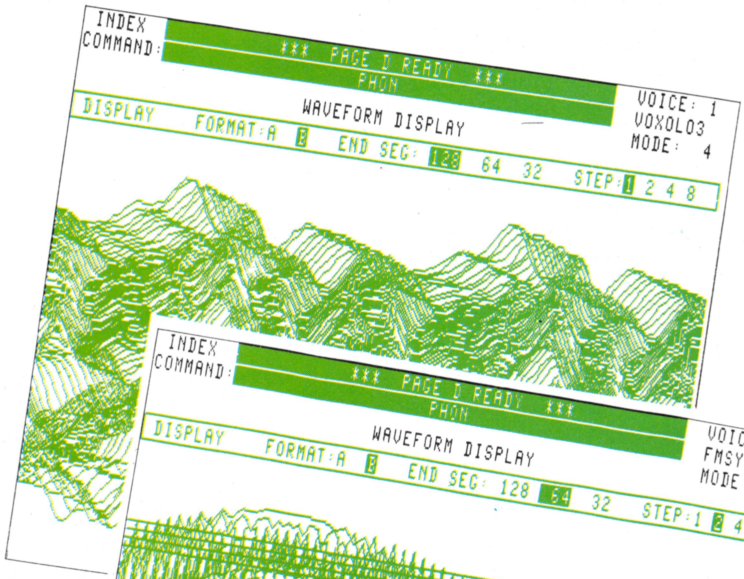
A system even more advanced than the Fairlight, Synclavier and Kurzweil combined is rumoured to be under development by Lucasfilm — the company responsible for *Star Wars*. Called the ASP (Audio Signal Processor), this is expected to incorporate every type of musical digital sound facility that is presently available only in a large music studio complex into one operating system. So, just as the computers and synthesisers of the 1950s took up space equivalent to the area of several rooms and yet now occupy only a desk-top, we can expect the recording studio of the future to be a portable package.

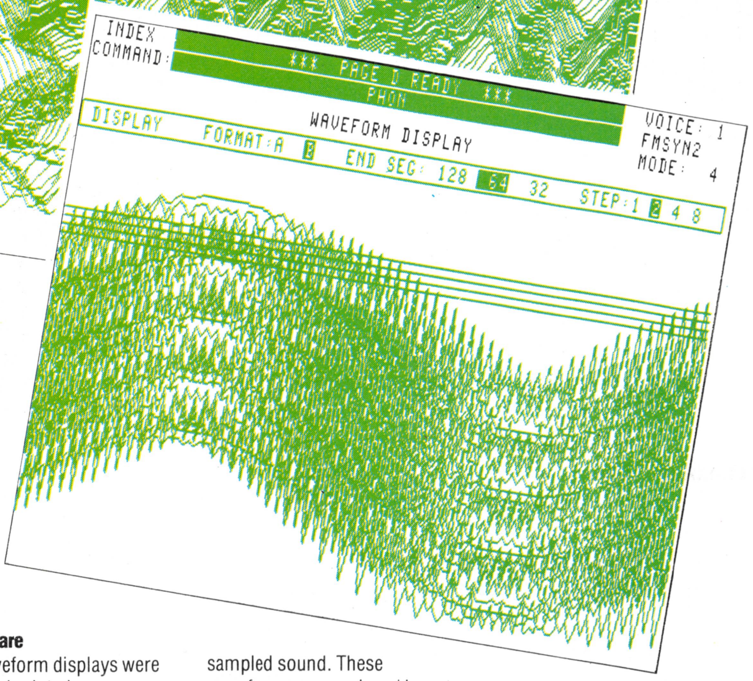
The incorporation of digital technology has not been confined solely to sound-generating systems and devices. Modern recording studios usually include a number of sound-treatment units as part of their basic equipment. An example is the *reverberation unit*. Music is routed through this


```
INDEX
COMMAND:
          PAGE 1... INDEX
          PAGE 2... DISK CONTROL
          PAGE 3... KEYBOARD CONTROL
          PAGE 4... HARMONIC ENVELOPES
          PAGE 5... WAVEFORM GENERATION
          PAGE 6... WAVEFORM DRAWING
Fairlight PAGE 7... CONTROL PARAMETERS
          PAGE 8... SOUND SAMPLING
          PAGE 9... KEYBOARD SEQUENCER
          PAGE A... ANALOG INTERFACE
          PAGE C... MCL COMPOSER
          PAGE D... WAVEFORM DISPLAY
          PAGE L... DISK LIBRARY
C.M.I.    PAGE R... REAL-TIME COMPOSER
V4:C5,R1:12 PAGE S... SCREEN PRINT
          USER NAME : PHIL NICHOLAS
```

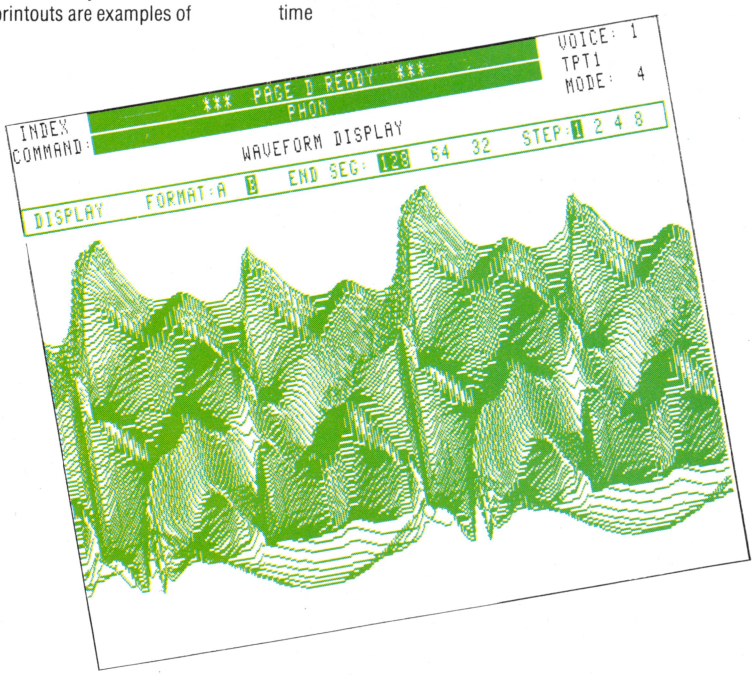
```
!B=24:G=1/24:V=15
R,36:<(Bf4:D4),5:(Bf4:D4),7
R,5:(Bf4:D4)V10,7/3:(Bf4:D4)V11,7/3:(F4:C4)V13,7/3
(Bf4:D4),5:(Bf4:D4),7
R,5:(Bf4:D4)V10,7/3:(Bf4:D4)V11,7/3:(F4:C4:Ef4)V13,7/3
(D4:F5),5:(D4:F5),7
R,5:(D4:F5)V10,7/3:(D4:F5)V11,7/3:(G4:B4:Ef4)V13,7/3
(Bf4:D4),5:(Bf4:D4),7
R,5:(Bf4:D4)V10,7/3:(Bf4:D4)V11,7/3:(F4:C4)V13,7/3>2
R,5:(Bf4:D4),7/3:(Bf4:D4),7/3:(F4:C4),7/3:T=+3>3:T=0
<(Bf4:D4),5:(C4:Ef4),7
<(Bf4:D4:A5),5:(C4:Ef4),7/2:(D4:Bf5),7/2>2
(Bf4:D4:A5),5:(Bf4:D4),7
(C4:Ef4:Bf5),5:(Bf4:D4),7/3:(Ef4:C4),7/3:(A5:C5),7/3
(C4:Ef4:Bf5),5:(Bf4:D4),7
(C4:Ef4:Bf5),5:(Bf4:D4),7/4:R,5/6
(Bf4:D4),7/8:R,5/12
(G4:B4:Ef4),75/24:V=15
<(Bf4:D4),5:(Bf4:D4),7
R,5:(Bf4:D4)V10,7/3:(Bf4:D4)V11,7/3:(F4:C4)V13,7/3
(Bf4:D4),5:(Bf4:D4),7
R,5:(Bf4:D4)V10,7/3:(Bf4:D4)V11,7/3:(F4:C4:Ef4)V13,7/3
(D4:F5),5:(D4:F5),7
R,5:(D4:F5)V10,7/3:(D4:F5)V11,7/3:(G4:B4:Ef4)V13,7/3
(Bf4:D4),5:(Bf4:D4),7
R,5:(Bf4:D4)V10,7/3:(Bf4:D4)V11,7/3:(F4:C4)V13,7/3>2
```

Fairlight MCL
The music composition language used by the Fairlight CMI is menu-driven, so choices are made from clear, thorough menus on the screen. The waveforms displayed on page 584 were generated with a list of sound parameters in data statements, similar to a BASIC listing, then displayed and printed out using commands from the menu


INDEX
COMMAND:
*** PAGE D READY ***
PHON
WAVEFORM DISPLAY
VOICE: 1
VOXOLO3
MODE: 4
DISPLAY FORMAT:A B END SEG: 128 64 32 STEP:1 2 4 8


INDEX
COMMAND:
*** PAGE D READY ***
PHON
WAVEFORM DISPLAY
VOICE: 1
FMSYN2
MODE: 4
DISPLAY FORMAT:A B END SEG: 128 64 32 STEP:1 2 4 8

Fairlight Fare
These waveform displays were created and printed on a Fairlight CMI music system. The first is an example of a sine wave pattern generated by the Fairlight using FM synthesis (page 561). This sound was fabricated by mixing waveforms electronically. The other printouts are examples of sampled sound. These waveforms are produced by digitising the actual sounds of a human voice, in the second instance, and a trumpet in the third. The displays are 'three dimensional', or topographical, representing the changes in the composition of each sound over time


INDEX
COMMAND:
*** PAGE D READY ***
PHON
WAVEFORM DISPLAY
VOICE: 1
TPT1
MODE: 4
DISPLAY FORMAT:A B END SEG: 128 64 32 STEP:1 2 4 8

unit to add echo or 'reverb', and rockabilly guitarists and dub reggae producers have depended on this type of treatment to give their music its particular sound.

The Quantec is a digital unit that, instead of merely providing reverberation, simulates the types of reverberation that occur in rooms and spaces of different sizes. Its smallest 'acoustic space' is a box with a volume of one cubic metre, and the preset simulations include sizes typical of living rooms, auditoriums, aircraft hangars and cathedrals. The Quantec's most interesting feature is the facility it offers to prolong the reverberation well beyond natural acoustic or physical limits. So, if a sound occurs in the cathedral-size simulation, and the reverberation time is maximised, the whole event will last several minutes — the effect is like listening to an echo provided by 10 Grand Canyons linked together!

It is said that, in the American Midwest in the 1950s, there was a small rockabilly studio that was built near a grain silo. It was this huge space that provided the studio with its classic echo-filled rockabilly sound. The modern use of digital units such as the Quantec is more mundane, being confined mainly to the post-production stages of television and film work. Actors can be filmed talking in the acoustically 'dead' environment of a studio, and the soundtrack can be treated afterwards to provide an acoustic resonance that suits the space in which the action is meant to occur.

'HUMAN' VALUES

Many people, musicians and non-musicians alike, feel that digital technology applied to music has a deleterious effect. They argue that this music is created and performed in such an artificial manner that the real 'human' values of spontaneity and expressiveness are being left behind in the rush to acquire greater degrees of technical control. This is a persuasive argument, but it is worth considering a visual medium — the cinema — to judge if this is really the case.

For several decades, cinema technology has allowed directors to shoot scenes from any angle, zoom in on detail and pan across large areas. It has been possible to repeat sequences of images, slow them down, speed them up, run them backwards, and edit film so that the most unlikely visual juxtapositions can occur within a split-second. It is only recently, with digital technology, that music-makers have begun to exercise comparable control over sound. Most people would accept cinema as an expressive medium, so it is likely that digital music will eventually be regarded in the same light. At present, it is easy for some musicians to be over-awed by the possibilities, and for others to produce music that has little more than digital novelty value. For those musicians prepared to meet the challenge of electronic music, the 1980s should be an exciting and demanding decade.

MOTOR CONTROL

In the last part of the Workshop series we looked at the construction of an interface that will enable us to control low voltage devices and considered the principles of motor control. Now we look at the software required to switch motors and develop a simple feedback control system.

Microprocessor control of external devices is now common in industry, its applications ranging from counting bottles on a conveyor belt to welding car bodies. The principles of all control systems are, firstly to input data from the outside world in a form acceptable to a microprocessor system; secondly to analyse that data; and thirdly to instigate external actions based on that analysis. If these three activities are repeated in a continuous cycle then we have a system known as *feedback control.*

To illustrate the principle of feedback control let us take the example of simmering a pan of soup on a stove. In order to cook the soup, the heat supplied must be sufficient to make it bubble, without making it boil over the lip of the pan. If we were to carry out this task ourselves, we might initially apply maximum heat to the pan until the soup started to bubble and then turn it down until the soup was simmering. If at any time the soup started to boil we would lower the heat further. In doing this sort of operation we monitor the state of the soup visually; analyse what we see; and take the appropriate action. We would repeat these actions until the soup was cooked. A microcomputer could control the cooking of the soup in a similar, but not identical way. The main differences would be in the way that the soup's state was monitored. Whereas we are able to look at the soup and make an assessment using our experience, a computer system would have to use another method based upon easily determined physical properties, such as temperature. Before the soup-monitoring was computerised, someone would need to carry out initial experiments to determine what temperature corresponded to an even simmer, and at which temperature the soup would boil over. From then on however, the computer could take over the job, provided that the appropriate devices were available to monitor temperature and regulate the heat.

Motors can be controlled in several ways using the buffer box (see page 546) and low voltage output box (see page 574) that we have constructed. A Lego or electric train set motor is ideal to test the software that we shall design. We need only to ensure that the motor we connect to the output box has a voltage rating equal to or greater than the input voltage from the transformer. Connecting the two motor terminals to line 0 of the output box, we can switch the motor on and off using the 'Z' and 'X' keys on the keyboard.

BBC MICRO

```
10 REM BBC SIMPLE MOTOR
20 DDR=&FE62:DATREG=&FE60
30 ?DDR=255:REM ALL LINES OUTPUT
40 ?DATREG=0:REM MOTOR OFF
50 REPEAT
60 A$=INKEY$(1):REM KEYPRESS?
70 IF A$="Z" THEN ?DATREG=1:REM TURN ON
80 UNTIL A$="X"
90 ?DATREG=0:REM TURN OFF
```

COMMODORE 64

```
10 REM CBM64 SIMPLE MOTOR
20 DDR=56579:DATREG=56577
30 POKEDDR,255: REM ALL LINES OUTPUT
40 POKEDATREG,0:REM MOTOR OFF
50 GETA$:IFA$<>"Z" ANDA$<> "X" THEN
   50:REM AWAIT KEY
60 IF A$="Z" THEN POKEDATREG,1:GOTO50
70 IF A$="X" THEN POKEDATREG,0:END
```

Exercises

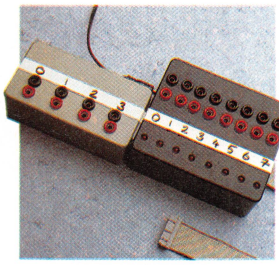
We can now design some elegant control experiments that rely on input and output using the buffer box and low voltage output box. The sensors referred to in the following experiments can be pressure sensors or 'reed' switches operated by a small magnet. Such devices are easily obtainable from electrical and electronics suppliers at low cost. Heat-sensitive switches operate by making an internal contact between the terminals when a certain temperature is reached and can be similarly obtained.

1) Write a program to make a vehicle run backwards and forwards between two sensors laid in its path.

2) Write a program to make a vehicle stop directly over a sensor, reversing back to it if necessary.

3) Write a program to keep a beaker of water between 70° and 100°C using a low voltage water heater and two heat-sensitive switches.

4) Write a program to calculate the speed in metres per second of a vehicle travelling between two points. (You will need to know the distance between the points and record the time taken to travel the distance.)


IAN McKINNELL

Put Out More Buffers
The buffer box, our first construction, is intended to protect the computer's circuitry from excessive input or output currents. Its independent power supply drives the output box — the more recent project — and allows software-controlled switching of the 12v output

We can control the direction of the motor by connecting the motor terminals to adjacent lines on the output box. The diagram shows these connections. We shall also use a simple make/break type switch, connected to line 7 of the buffer box to control the direction.

In the following program a value of 1 in the data register causes current to flow one way through the motor. Placing a value of 2 in the data register will cause the current to flow in the reverse direction. The program repeatedly tests line 7 and only places a 2 in the data register when the line is set low (i.e. the switch is closed). In this way, the closing and opening of the switch controls the motor's direction. This is a very simple example of a feedback control system.

BBC MICRO

```
10 REM BBC DIRECTED MOTORS
20 DDR=&FE62:DATREG=&FE60
30 ?DDR=127:REM LINE 7 INPUT
40 ?DATREG=0:REM TURN OFF
50 A$=GET$:REM AWAIT KEYPRESS
60 REPEAT
70 A$=INKEY$(1)
80 IF (?DATREG AND 128)=0 THEN DIR=2 ELSE DIR=1
90 ?DATREG=DIR
100 UNTIL A$="X":REM PRESS X TO END
110 ?DATREG=0:REM TURN OFF
```

COMMODORE 64

```
10 REM CBM64 DIRECTED MOTORS
20 DDR=56579:DATREG=56577
30 POKEDDR,127:REM LINE 7 INPUT
40 POKEDATREG,0:REM ALL OFF
50 GETA$:IFA$=" "THEN50:REM AWAIT KEYPRESS
60 GETA$
70 IF (PEEK(DATREG)AND128)=0THENPOKEDATREG,
   2:GOTO90
80 POKEDATREG,1
90 IFA$<>"X"THEN60
100 POKEDATREG,0:REM TURN OFF
```

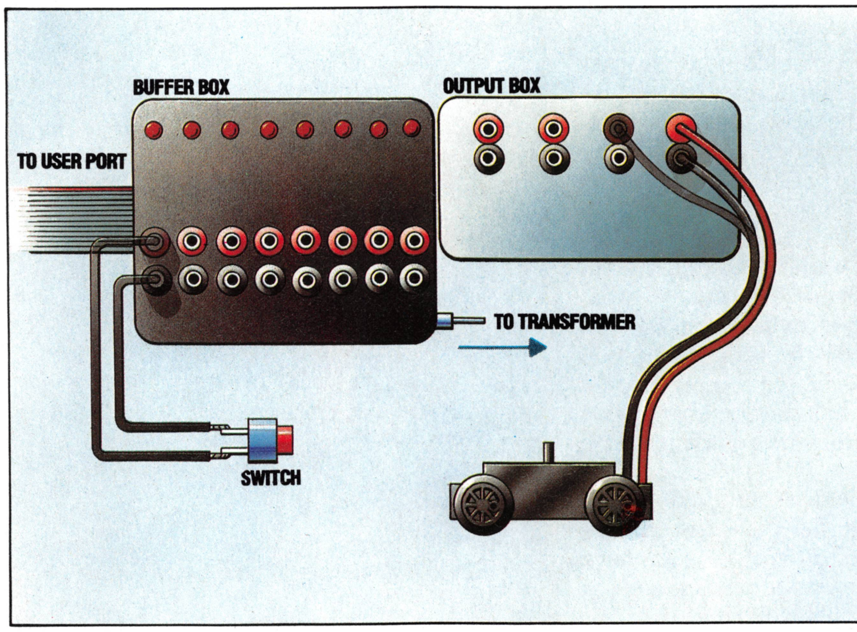
In addition to being able to control the direction of motors, we can also control their speed directly from the output box. This does not require complicated devices, such as digital-to-analogue converters to control the supply to the motors. Instead we can send pulses to the motor, turning it on and off in rapid succession. If we do this fast enough, the motor will appear to rotate continuously; the interval between each pulse determining the speed at which the motor turns. In order to program this, all we require is a pair of delay loops of adjustable length, within a larger repetitive structure, to determine the length of time that the motor is on and off during each cycle.

BBC MICRO

```
10 REM BBC VARIABLE MOTOR CONTROL
20 DDR=&FE62:DATREG=&FE60:SPEED=30
30 ?DDR=255:REM ALL OUTPUT
40 ?DATREG=0:REM ALL OFF
50 A$=GET$;REM AWAIT KEYPRESS
60 REPEAT
70 A$=INKEY$(1)
80 ?DATREG=0:REM TURN OFF
90 FORI=1TO(100-SPEED):NEXT:REM DELAY1
100 ?DATREG=1:REM TURN ON
110 FORI=1TO SPEED:NEXT:REM DELAY 2
120 IF A$="D"THEN SPEED=SPEED-5
130 IF A$="Z"THEN SPEED=SPEED+5
140 UNTIL A$="X"
150 ?DATREG=0:REM TURN OFF
```

COMMODORE 64

```
10 REM CBM64 VARIABLE MOTOR CONTROL
20 DDR=56579:DATREG=56577:SPEED=30
30 POKEDDR,255:REM ALL LINES OUTPUT
40 POKEDATREG,0:REM TURN OFF
50 GETA$:IFA$=" "THEN50:REM AWAIT KEY
60 GETA$
70 POKEDATREG=0:REM TURN OFF
80 FORI=1TO(100-SPEED):NEXT:REM DELAY1
```



Four Wheel Drive
The two boxes share the data bus and power supply through their minicon ports; if each box is built with a socket and a plug, then boxes can be connected in chains, 'piggyback' fashion.

The car has one 12v DC motor, supplied from the output box, and switched by bit 7 of the buffer box. The polarity of the outputs can be reversed under software control, driving the car forward and backward

KEVIN JONES

```
90 POKEDATREG=1:REM TURN ON
100 FORI=1TOSPEED:NEXT:REM DELAY2
110 IFA$="D"THENSPEED=SPEED-5
120 IFA$="Z"THENSPEED=SPEED+5
130 IFA$<>"X"THEN60
140 POKEDATREG,0:REM TURN OFF
```

In this program the variable SPEED is used to determine the length of each delay loop. The loop code is such that as one delay increases, the other decreases and vice-versa. DELAY 1 determines the period when the motor is off and DELAY 2 the period when the motor is on. For large values of SPEED, the first delay is short and the second is long, making the motor turn more quickly. Smaller values of SPEED will produce longer periods when the motor is off during each cycle, making the motor appear to turn more slowly. The pulsing effect that the program has can be observed in the flicker of line 1's LED.

So Far, So Good
Driving wire guided toy cars around on the floor may not seem an enormous return on the effort and capital invested, but building and programming even these simple artefacts has introduced us to the reality of electronic and electromechanical construction, and demonstrated some of the problems and opportunities involved in making software interact with the real world

LEGO CONSTRUCTION BY DAVE WHELAN
IAN McKINNELL

Exercise Answers

1) Assuming that the sensors are connected to lines 6 and 7, and the motor is connected between the positive terminals of lines 0 and 1:

```
10 REM BBC VERSION 3.1
20 DDR=&FE62:DATREG=&FE60
30 ?DDR=63:REM LINES 6&7 INPUT
40 forward:1:reverse=2
50 ?DATREG=forward
60 FORI=1TO2000:NEXT:REM DELAY
70 REPEAT UNTIL(?DATREG AND192)<>192
80 ?DATREG=reverse
90 FORI=1TO2000:NEXT REM DELAY
100 IF(?DATREG AND192)<>192 THEN50
    ELSE GOTO100
```

```
10 REM CBM 64 VERSION 3.1
20 DDR=56579:DATREG=56577
30 POKEDDR,63:REM LINES 6&7 INPUT
40 FW=1:RV=2
50 POKE DATREG,FW
60 FORI=1TO1000:NEXT:REM DELAY
70 IF(PEEK(DATREG)AND192)=192THEN70
80 POKE DATREG,RV
90 FORI=1TO1000:NEXT:REM DELAY
100 IF(PEEK(DATREG)AND192)<>192THEN50
110 GOTO100
```

2) Assuming that the sensor connects to line 7 and the motor connects between lines 0 and 1:

```
10 REM BBC VERSION 3.2
20 DDR=&FE62:DATREG=&FE60
30 ?DDR=127:REM LINE 7 INPUT
40 speed=30:forward=1:reverse=2
50 direction=forward
60 REPEAT
70 ?DATREG=0:REM OFF
80 FORI=1TO(100-speed):NEXT
90 ?DATREG=direction
100 FORI=1TOspeed:NEXT
110 UNTIL(?DATREG AND128)=0:REM SWITCH
120 FORI=1TO1000:NEXT:REM DELAY
130 REM TEST FOR OVER PAD
140 IF(?DATREG AND128)=0THEN?DATREG=0:
    END
150 REM REVERSE SLOWLY
160 speed=2:direction=reverse:GOTO60
```

```
10 REM CBM 64 VERSION 3.2
20 DDR=56579:DATREG=56577
30 POKE DDR,127:REM LINE 7 INPUT
40 SP=30:FW=1:RV=2
50 DR=FW
60 POKE DATREG,0: REM OFF
70 FORI=1TO(100-SP):NEXT
80 POKE DATREG,DR
90 FORI=1TOSP:NEXT
100 IF(PEEK(DATREG)AND128)<>0THEN60
110 FORI=1TO1000:NEXT:REM DELAY
120 IF(PEEK(DATREG)AND128)=0THENPOKE
    DATREG,0:END
130 REM REVERSE BACK SLOWLY
140 SP=2:DR=RV:GOTO60
```

3) Assuming that the 40 and 70 degree sensors are connected to lines 6 and 7 respectively, and the heater is attached to line 0:

```
10 REM BBC VERSION 3.3
20 DDR=&FE62:DATREG=&FE60
30 ?DDR=63:REM LINES 6&7 INPUT
40 REPEAT
50 A$=INKEY$(1)
60 ?DATREG=1:REM TURN ON HEATER
70 REPEAT
80 UNTIL(?DATREG AND192)=0:REM 70 DEG
90 ?DATREG=0:REM HEATER OFF
100 REPEAT UNTIL(?DATREG AND192)=192
110 UNTIL A$<>"":REM KEYPRESS
```

```
10 REM CBM 64 VERSION 3.3
20 DDR=56579:DATREG=56577
30 POKE DRR,63:REM LINES 6&7 INPUT
40 GET A$
50 ?DATREG=1:REM TURN HEATER ON
60 IF(PEEK(DATREG)AND192)<>0THEN60
70 POKE DATREG,0:REM HEATER OFF
80 IF(PEEK(DATREG)AND192)<>192THEN80
90 IF A$=""THEN40
```

4) Assuming that the first switch is on line 6 and the second is on line 7:

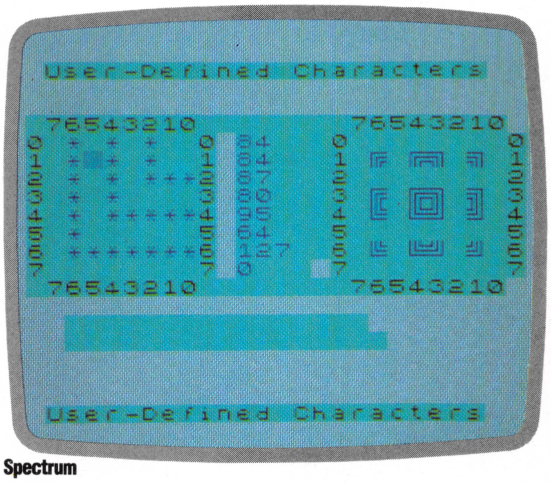
```
10 REM BBC VERSION 3.4
20 DDR=&FE62:DATREG=&FE60:DISTANCE=1
30 ?DDR=63
40 REPEAT UNTIL(?DATREG AND64)=0
50 ?DATREG=1
60 TIME=0:REM START TIMER
70 REPEAT UNTIL(?DATREG AND128)=0
80 PRINT"SPEED="DISTANCE/(TIME/100)
90 ?DATREG=0
```

```
10 REM CBM 64 VERSION 3.4
20 DDR=56579:DATREG=56577:DS=1
30 POKE DDR,63
40 IF(PEEK(DATREG)AND64)<>0THEN40
50 POKE DATREG,1
60 T=TI:REM START TIMER
70 IF(PEEK(DATREG)AND128)<>0THEN60
80 PRINT"SPEED="DS/((TI-T)/60)
90 POKE DATREG,0
```

CHARACTER FORMING

We have looked at how to define your own characters on the Commodore, here we provide two programs for the BBC Micro and the Spectrum. Both machines have an area of memory reserved for the user's characters, which makes character generation a far simpler exercise.

The Spectrum's approach to user-defined characters is typically matter-of-fact: 168 bytes at the top of memory are dedicated by the operating system to the user's character definitions (though this memory can be used for other purposes, such as storing machine code programs or data). This space allows 21 character definitions to be stored, and the operating system attaches a CHR$ value in the range 144 to 164 to each definition. The characters so defined are regarded by the machine as the characters 'a' to 'u' in graphics (G) mode. The user-defined graphics (UDG) system variable (address 32600) points to the first byte of the dedicated memory area, but you don't have to do complicated sums with this variable to find the start address of a character definition. The command LET address = USR "A" returns the address of the first byte of the definition of the character inside the quotes.



Spectrum

Comparing the length and structure of the Commodore and Spectrum lists demonstrates the relative ease of use of the Spectrum's methods.

The program is a straightforward translation of the Commodore version (see page 572). The Spectrum's normal cursor keys (shifted 5, shifted 6, etc.) control the edit cursor, and the unshifted 6, 7 and 8 keys are the command keys — toggle a cell, change the edit character, and place a character in the text window. The exclamation mark is the exit command.

When you have defined your new characters, you can SAVE the UDG area by the following command:

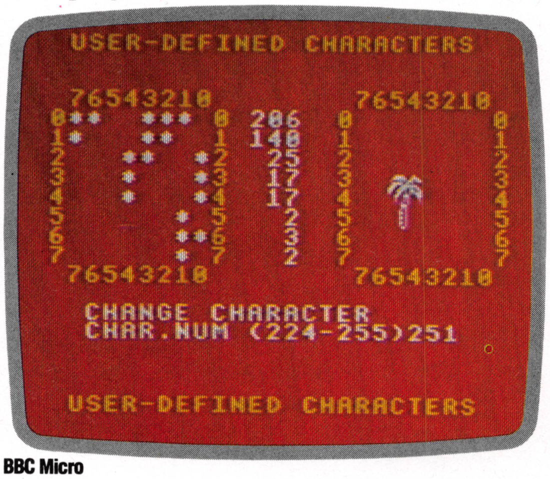
```
SAVE "udgfile" CODE (USR "A"), 168
```

and reload it thus:

```
LOAD "udgfile" CODE
```

The BBC Micro's treatment of user-defined graphics is similar, at first glance, to the Spectrum's. The 256 bytes between &0C00 and &0CFF are reserved for the definitions of the ASCII-coded characters 224 to 255. If the character set is imploded (see *The BBC Advanced User Guide*, page 136) then those definitions are also applied to the ASCII codes 128 to 159, 160 to 191 and 192 to 223. In the exploded state, the entire printable character set (from CHR$(32) to CHR$(255)) can be redefined, but at the cost of user RAM. For most purposes, a maximum of 32 special characters should suffice.

The program functions are the same as in the Commodore and Spectrum versions. The arrow keys control the cursor, and the function keys, f1 to f3, access the toggle, redefine and place commands. As before, the exclamation mark, '!', is the program terminator.



BBC Micro

IAN McKINNELL

Notice the OS calls at lines 70 and 180: these disable and enable the COPY functions so that the arrow keys can be used in the program; if you quit the program other than through the '!' command, you must type *FX4,0 at the start of a new line to re-enable the editing keys.

The BBC Micro does allow you to use VDU23 to define characters, but in this case it is simpler to calculate the relevant addresses and then PEEK and POKE them individually.

You can SAVE your special characters thus:

```
*SAVE "filename' 0C00 0CFF
```

and reload them thus:

```
*LOAD""
```

Spectrum

```
  19 REM **********************
  20 REM *      spectrum      *
  21 REM *usr defined char-gen*
  22 REM **********************
 100 GO SUB 1000: REM initialise
 110 FOR j=0 TO 1 STEP 0
 120 GO SUB 2500: REM input
 130 GO SUB 3000: REM validate
 140 GO SUB pointer: REM obey
 150 NEXT j
 200 STOP
 999 REM **********************
1000 REM *   initialise       *
1001 REM **********************
1020 DIM b(8,8):   DIM c$(2,2):
 DIM o(2,7): DIM r$(8): DIM d$(
1): DIM t(8,8)
1100 LET nullkey=2000: LET movec
rsr=3500: LET command=3000: LET
update=6500
1200  REM ****init screen****
1220 LET r0=4: LET c0=3: LET rl=
8: LET cl=8: LET of=16
1230 LET black=0: LET blue=1: LE
T cyan=5: LET white=7
1240 PAPER cyan: INK black
1250 LET z$="User-Defined Charac
ters": PRINT AT 1,4;z$: PRINT AT
 20,4;z$
1260 LET z$=" 76543210": PRINT A
T r0,c0;z$;: PRINT TAB c0+of;z$
1270 PRINT AT r0+cl+1,c0;z$;: PR
INT TAB c0+of;z$
1280 FOR r=r0+1 TO r0+cl
1290 LET z$=STR$ (r-r0-1): LET z
$=z$+"        "+z$
1300 PRINT AT r,c0;z$;AT r,c0+of
;z$
1310 NEXT r
1420 REM ****cursor offset*****
1440 DATA -1,+1,0,0
1445 DATA 0,0,+1,-1
1450 DATA 0,0,+1,-1
1460 FOR k=1 TO 2: FOR l=1 TO 4
1470 READ o(k,l): NEXT l: NEXT k
1600 REM ***init text window***
1620 FOR r=1 TO rl
1640 FOR c=1 TO cl
1660 LET t(r,c)=32
1680 NEXT c: NEXT r
1800 LET rpos=1: LET cpos=1: LET
 cn=144: LET e$="A"
1820 GO SUB 6000
1990 RETURN
1999 REM **********************
2000 REM * invalid keypress   *
2001 REM **********************
2020 BEEP .2,-3: RETURN
2120 RETURN
2499 REM **********************
2500 REM * crsr @ rpos,cpos   *
2501 REM **********************
2520 LET rp=r0+rpos: LET  cp=c0+
cpos: LET cf=1+b(rpos,cpos): LET
 d$=" *"(cf)
2540 PRINT AT rp,cp; FLASH 1;d$
2600 REM  LET t$=INKEY$: IF t$<>
"" THEN  GO TO 2600
2620 LET t$=INKEY$: IF t$="" THE
N  GO TO 2620
2640 PRINT AT rp,cp; FLASH 0;d$
2700 RETURN
2999 REM **********************
3000 REM * validate input     *
3001 REM **********************
3020 IF t$="!" THEN  LET pointer
=nullkey: LET j=2: RETURN
3040 LET key=CODE (t$)-7: LET k=
key-45: LET pointer=nullkey
3060 IF (key>=1 AND key<=4) THEN
  LET pointer=movecrsr: RETURN
3080 IF (t$>="6" AND t$<="8") TH
EN  LET key=VAL (t$)-4 :    LET
  pointer=command+key*500: RETUR
N
3100 RETURN
3499 REM **********************
3500 REM * move the cursor    *
3501 REM **********************
3520 LET ny=rpos+o(2,key):  LET
nx=cpos+o(1,key)
3540 IF  (ny<1 OR ny>rl)  THEN
  RETURN
3560 IF  (nx<1 OR nx>rl)  THEN
  RETURN
3580 LET rpos=ny: LET cpos=nx
3600 RETURN
3999 REM **********************
4000 REM * toggle a cell      *
4001 REM **********************
4020 LET tg=1-b(rpos,cpos): LET
d$=" *"(1+tg)
4040 PRINT AT rpos+r0,cpos+c0;d$
4060 LET b(rpos,cpos)=tg
4120 LET pe=PEEK (mpos+rpos)
4140 LET pe=pe+(tg*2-1)*(2^(8-cp
os))
4160 POKE (mpos+rpos),pe
4200 PRINT AT r0+rpos,c0+cl+3;PE
EK (mpos+rpos);"  "
4220  NEXT r
4300 RETURN
4499 REM **********************
4500 REM * define new char    *
4501 REM **********************
4520 PRINT  AT  r0+rl+3,c0+2;
 FLASH 1;"Change Character"
4530 GO SUB update
4540 PRINT  AT  r0+rl+4,c0+2;
;"Hit A Key (a - u)"
4550 FOR z=1 TO 1
4560 LET e$=INKEY$: IF e$<>""
 THEN  GO TO 4560
4570 LET e$=INKEY$: IF e$=""
THEN  GO TO 4570
4580 IF (e$<"a" OR e$>"u") THEN
 GO SUB nullkey: LET z=0
4590 NEXT z
4600 LET cn=CODE (e$)+79
4610 IF cn>164 THEN  LET cn=cn-3
2
4620 PRINT  AT  r0+rl+3,c0+2;
 FLASH 0;"                "
4630 PRINT AT  r0+rl+4,c0+2;
"                 "
4640 GO SUB 6000
4990 RETURN
4999 REM **********************
5000 REM * place a char       *
5001 REM **********************
5020 PRINT AT rpos+r0,cpos+c0+
of;CHR$ (cn)
5040 LET t(rpos,cpos)=cn
5490 RETURN
5999 REM **********************
6000 REM *  display a char    *
6001 REM **********************
6020 LET mpos=USR (e$)-1:
 INK blue
6040 FOR r=1 TO cl: LET pe=PEEK
(mpos+r): LET p=pe: LET z$=""
6060 FOR c=cl TO 1 STEP -1: LET
n=INT (pe/2): LET q=pe-2*n
6080 LET b(r,c)=q: LET pe=n
6100 LET r$(c)=" *"(q+1): NEXT c
6120 PRINT AT r0+r,c0+1;r$;AT r0
+r,c0+cl+3;p;"  "
6130 NEXT r
6140 RETURN
6499 REM **********************
6500 REM * update text window *
6501 REM **********************
6520 FOR r=  TO 8: LET y=r0+r
6540 FOR c=1 TO 8: LET x=c0+c+of
6560 PRINT AT y,x;CHR$ (t(r,c))
6580 NEXT c: NEXT r
6600 RETURN
```

BBC Micro

```
  19 REM***********************
  20 REM*     BBC Micro       *
  21 REM***********************
  22 REM* USER-DEF CHGEN      *
  50 MODE 1
  60 @%=3
  70 *FX4,1
 100 PROCinitialise
 110 REPEAT
 120 PROCget_command
 130 PROCvalidate_command
 140 PROCobey(COMMAND)
 160 UNTIL COMMAND=QUIT
 180 *FX4,0
 200 END
 999 REM***********************
1000 DEFPROCinitialise
1001 REM***********************
1040 DIM BD(8,8),C$(2,2)
1045 DIMOFST(2,7),D$(1),TX(8,8)
1060 D$(0)=" ":D$(1)="*"
1080 CGEN=&0C00-1
1100 QUIT=-1:NULLKEY=0:MVCRSR=1:
1105 TGGLE=2:DFINE=3:PLACE=4
1200 REM*****INIT SCREEN*****
1220 R0=4:C0=3:RL=8:CL=8
1225 C9=CL+3:OF=16
1230 BLACK=0:RED=1:YELLOW=2
1235 WHITE=3:BACKGR=128
1240 COLOUR BACKGR+RED
1245 COLOUR YELLOW:CLS
1250 Z$="USER-DEFINED CHARACTERS"
1255 PRINTTAB(4,1)Z$;TAB(4,20)Z$
1260 Z$=" 76543210"
1264 PRINTTAB(C0,R0)Z$
1268 PRINTTAB(C0+OF,R0)Z$
1270 Y=R0+RL+1:X=C0
1274 PRINTTAB(X,Y)Z$
1278 PRINTTAB(X+OF,Y)Z$
1280 FOR R=R0+1 TO R0+RL
1290 Z$=STR$(R-R0-1)
1295 Z$=Z$+"        "+Z$
1300 PRINTTAB(C0,R)Z$
1304 PRINTTAB(C0+OF,R)Z$
1310 NEXT R
1320 COLOUR WHITE
1420 REM****CURSOR OFFSET****
1440 DATA -1,+1,0,0
1450 DATA 0,0,+1,-1
1460 FOR K=1 TO 2:FOR L=1 TO 4
1470 READ OFST(K,L):NEXT L,K
1500 REM****KEY SETTINGS*****
1520 *KEY1"T"
1540 *KEY2"D"
1560 *KEY3"P"
1600 REM***INIT TEXT WINDOW**
1620 FOR R=1 TO RL
1640 FOR C=1 TO P!.
1660 TX(R,C)=32
1680 NEXT C,R
1800 RPOS=1:CPOS=1:CN=224
1805 PROCdisplay_character
1990 ENDPROC
1999 REM***********************
2000 DEFPROCnullkey
2001 REM***********************
2020 SOUND 1,-15,48,10
2120 ENDPROC
2499 REM***********************
2500 DEFPROCget_command
2501 REM***********************
2520 RP=R0+RPOS:CP=C0+CPOS
2540 PRINTTAB(CP,RP);
2620 T$=GET$
2700 ENDPROC
2999 REM***********************
3000 DEFPROCvalidate_command
3001 REM***********************
3020 COMMAND=MVCRSR
3040 KEY=ASC(T$)-135
3060 IF KEY<1 THEN COMMAND=NULLKEY
3065 IF KEY>4 THEN COMMAND=NULLKEY
3080 IF T$="T"THEN COMMAND=TGGLE
3100 IF T$="D"THEN COMMAND=DFINE
3120 IF T$="P" THEN COMMAND=PLACE
3150 IF T$="!" THEN COMMAND=QUIT
3160 ENDPROC
3299 REM***********************
3300 DEFPROCobey(command)
3301 REM***********************
3320 IF command=QUIT THEN
PROCnullkey
3340 IF command=MVCRSR THEN
PROCmovecursor(KEY)
3360 IF command=TGGLE THEN
PROCtoggle_a_cell
3380 IF command=DFINE THEN
PROCdefine_character
3400 IF command=PLACE   THEN
PROCplace_character
3420 IF command=NULLKEY THEN
PROCnullkey
3450 ENDPROC
3499 REM***********************
3500 DEFPROCmovecursor(key)
3501 REM***********************
3520 NY=RPOS+OFST(2,key)
3525 NX=CPOS+OFST(1,key)
3540 IF NY>=1 AND NY<=RL THEN
RPOS=NY ELSE PROCnullkey
3560 IF NX>=1 AND NX<=CL THEN
CPOS=NX ELSE PROCnullkey
3600 ENDPROC
3999 REM***********************
4000 DEFPROCtoggle_a_cell
4001 REM***********************
4020 TG=1-BD(RPOS,CPOS)
4025 Y=RPOS+R0:X=CPOS+C0
4040 PRINTTAB(X,Y)D$(TG)
4060 BD(RPOS,CPOS)=TG
4120 PE=?(MPOS+RPOS)
4140 PE=PE+(TG*2-1)*(2^(8-CPOS))
4160 ?(MPOS+RPOS)=PE
4200 PRINTTAB(C0+C9,R0+RPOS)PE
4220 PROCupdate_text
4300 ENDPROC
4499 REM***********************
4500 DEFPROCdefine_character
4501 REM***********************
4510 REPEAT
4520 PRINTTAB(C0+2,R0+RL+3);
4525 PRINT"                      "
4540 PRINTTAB(C0+2,R0+RL+4);
4545 PRINT"                      "
4550 PRINTTAB(C0+2,R0+RL+4);
4560 INPUT"NUMBER (224-255)"CHNUM
4580 UNTIL CHNUM>223 AND CHNUM<256
4600 CN=CHNUM
4620 PROCdisplay_character
4900 ENDPROC
4999 REM***********************
5000 DEFPROCplace_character
5001 REM***********************
5015 X=CPOS+C0+OF:Y=RPOS+R0
5020 PRINTTAB(X,Y);CHR$(CN)
5040 TX(CPOS,RPOS)=CN
5400 ENDPROC
5999 REM***********************
6000 DEFPROCdisplay_character
6001 REM***********************
6020 MPOS=(CN-224)*8+CGEN
6040 FOR R=1 TO CL
6045 PE=?(MPOS+R):P=PE:Z$=""
6060 FOR C=CL TO 1 STEP -1
6065 N=INT(PE/2):Q=PE-2*N
6080 BD(R,C)=Q:PE=N
6100 Z$=D$(Q)+Z$:NEXT C
6120 PRINTTAB(C0+1,R0+R)Z$
6125 PRINTTAB(C0+C9,R0+R)P
6130 NEXT R
6140 ENDPROC
6499 REM***********************
6500 DEFPROCupdate_text
6501 REM***********************
6520 FOR R=1 TO RL:Y=R0+R
6540 FOR C=1 TO RL:X=C0+C+OF
6560 PRINTTAB(X,Y)CHR$(TX(C,R))
6580 NEXT C,R
6900 ENDPROC
```

HOT PROPERTY

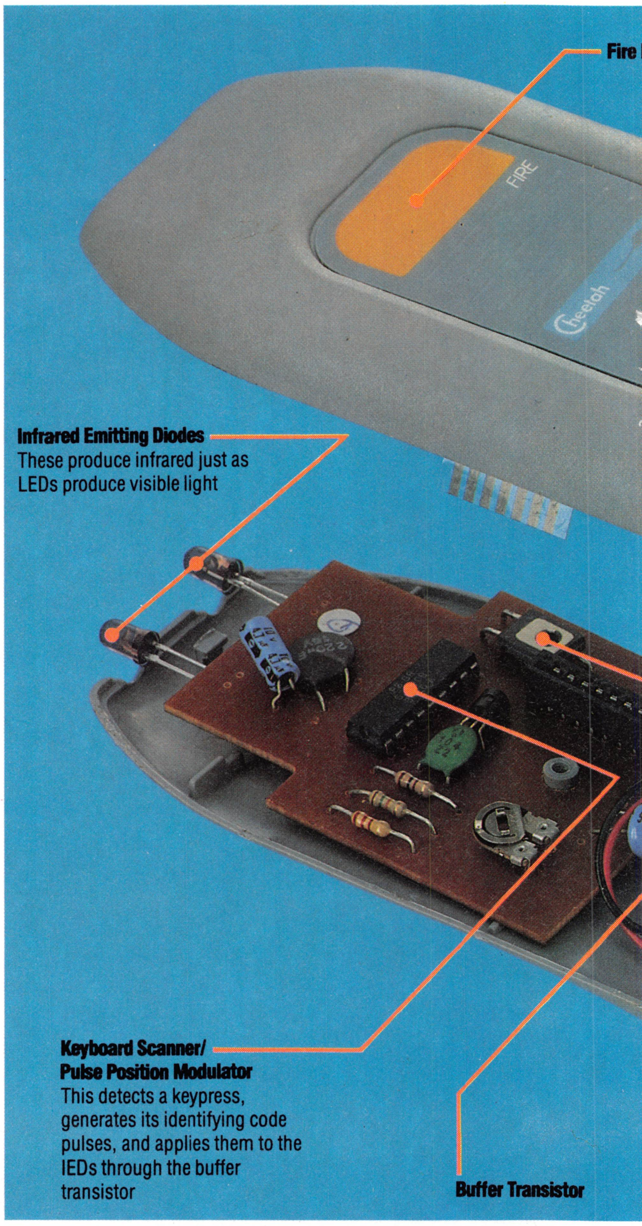
Cheetah Marketing is a company with a good track record in producing add-ons for the Sinclair Spectrum — the Sweet Talker synthesiser is the most successful to date. The RAT, a remote-controlled joypad that uses infrared waves, is the company's latest achievement. We look at this addition to the home computer menagerie...

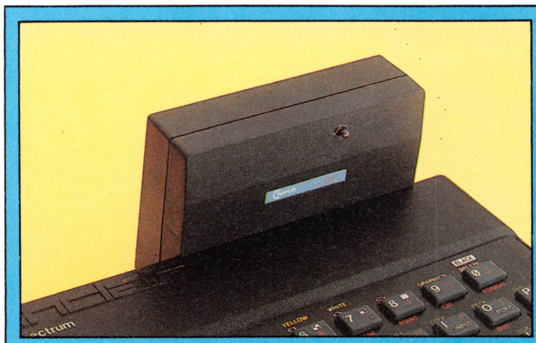
Serious arcade games players tend to be very concerned about anything that significantly affects the speed and quality of their play — especially if it has a noticeable impact on the final score. For this reason, the way in which a game is controlled is a matter of great importance. With games controlled from the keyboard, the major concerns are the choice of control keys and the ease with which these can be used. Consequently, software writers must pay particular attention to this sort of detail. With external controllers, like joysticks, the design of the hardware tends to be the crucial factor.

The flexibility of a joystick — its freedom of movement, how quickly it reacts to your touch, the speed at which the game responds — is its most significant design consideration. But the designer is often hampered by the limitations imposed by the nature of a joystick itself — the length of the connecting lead, the size, shape and position of the controller, and the position of the fire button(s). The latter detail, for example, often favours right-handed players. Although joystick manufacturers have tried to develop designs that overcome some of these drawbacks, none have been as successful as Cheetah Marketing's infrared remote joypad for the Sinclair Spectrum.

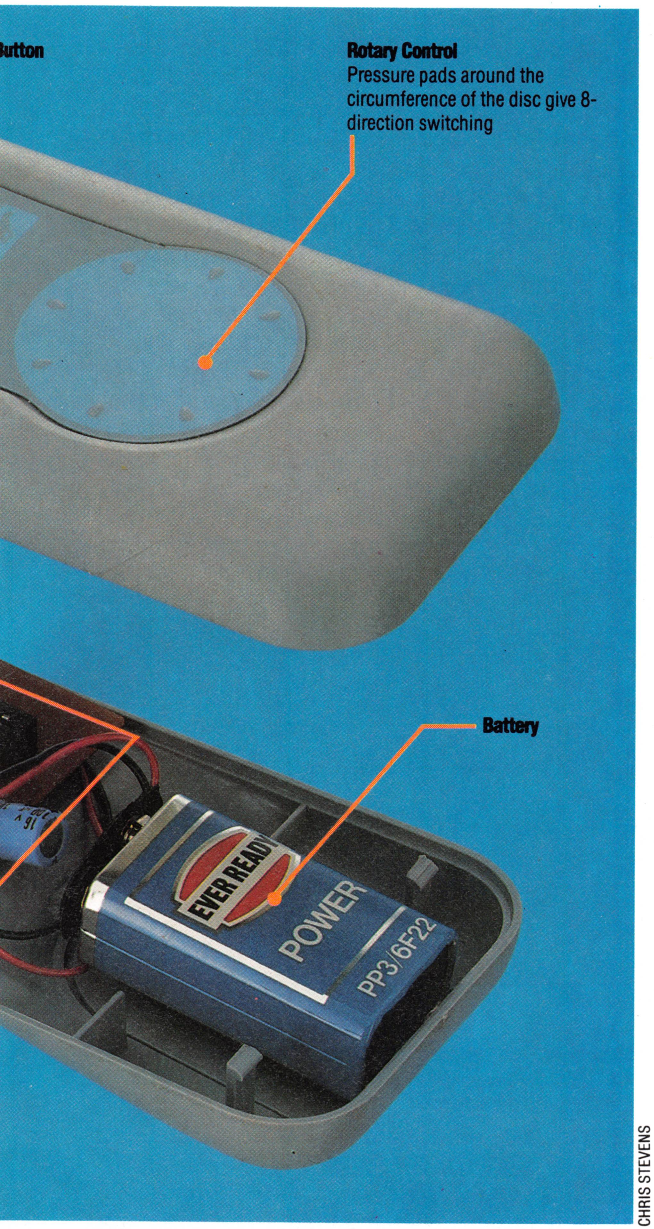
Cheetah has called its joypad the RAT. The name is said to be an abbreviation for 'remote action transmitter', but seems to be a play on the word Mouse, which is applied to the hand-held controllers used with Apple's Macintosh and other machines. The RAT looks like a slightly elongated phaser weapon from television's *Star Trek* series. It is long, flat and grey, with a blue circular control pad, the Cheetah logo and a bright orange fire button. There are two infrared transmitters extending from the front of the unit. When you first hold the RAT and press the fire button you almost expect flashes of blue flame to leap from it.

The system also includes its own interface, which plugs into the edge connector at the back of the Spectrum. This box has an expansion port of its own for further add-ons. There is a single


Fire
FIRE
Cheetah
Infrared Emitting Diodes
These produce infrared just as LEDs produce visible light
Keyboard Scanner/ Pulse Position Modulator
This detects a keypress, generates its identifying code pulses, and applies them to the IEDs through the buffer transistor
Buffer Transistor


CHRIS STEVENS

Radiating Light
Infrared radiation — having a longer wavelength than visible light, but shorter than radio waves — is produced in the transmitter by Infrared Emitting Diodes (IEDs) when an electrical current in a tiny chip of gallium arsenide excites the molecules causing photons to be released. In the receiver, conversely, an electrical current flows in the IED when infrared light falls upon the gallium arsenide. When the control buttons on the RAT are pressed, therefore, the two transmitter IEDs emit coded pulses of infrared in a broad beam, triggering the receiver directly or after reflection in the room


Button
Rotary Control
Pressure pads around the circumference of the disc give 8-direction switching
Battery
EVER READY
POWER
PP3/6F22
CHRIS STEVENS

Mixed Reception
The original keyboard of the long-awaited IBM PC Junior was noted mainly for the low quality of its appearance and engineering, but it was also the first microcomputer with infrared linking between keyboard and processor

infrared receiver on the front of this unit for communicating with the RAT.

The package has a sheet of instructions that explain how the joypad is used and what games can be played with it (any software that is Kempston joystick-compatible). With great foresight, Cheetah has included routines in BASIC and machine code that enable you to incorporate RAT control in your own games.

The instruction sheet claims that the RAT can be used at distances up to 12 feet by aiming 'in the general direction of the computer'. Movement is effected by pressing lightly on the blue control pad. There are eight small 'bumps' on the periphery of the pad, and pressing on or near one of these indicates the direction required — N, SW, and so on, rather like the directions on a compass. While one hand holds the RAT and controls the direction of movement on the screen, the other hand can be used to press the fire 'button'. Because of the design of the RAT, it makes no difference which hand performs each task, so the joypad works equally well for left- and right-handed players. The transmitter requires one PP3 battery, which fits into a small space at the back of the unit directly below the blue control circle.

Once the interface box is connected, the transmitter unit has been fitted with a battery and a game requiring joysticks has been loaded into your Spectrum, then you are ready to play. Because there is no visible sign that the transmitter is working until you can see movement on the screen, you might find yourself sticking as close to the computer as you would with any other joystick. There is a certain reluctance to believe the claims of a 12-foot control distance. But when you realise that the RAT really works, you will want to experiment to see how far it can go.

In fact, the remote action transmitter works extremely well at distances even slightly over 12 feet. And it doesn't have to be pointed in the general direction of the computer. The RAT works when pointed straight up at the ceiling, down at the floor, over your back or sideways (although it is slightly difficult for you to see what you are doing when the transmitter is pointed at odd angles). Clearly, the Cheetah RAT gives a games player a tremendous freedom to move around. The biggest drawback, however, is that it has only eight positions of movement — up, down, right, left, and intermediate points. It would be better to have more control than this.

The fact that the RAT has no moving parts makes the unit less prone to wear and tear than standard joysticks, so it should last a long time. In fact, it comes with a year's warranty. At £29.95, the Cheetah RAT costs only slightly more than most other joysticks plus Interface 2. (Of course, if you already have Interface 2, this is not much consolation.) But its play allows much more freedom of movement and far better control than most joysticks.

CHEETAH REMOTE ACTION TRANSMITTER INFRARED JOYPAD

FOR: Sinclair ZX Spectrum

PRICE: £29.95, including interface box

WORKS WITH: Cheetah compatible or Kempston joystick compatible games

FROM: Cheetah Marketing,Ltd, 24 Roy St, London EC1

F

FILE PROTECTION

Computer files stored on tape or disk can generally be accessed by anyone who knows the name of a particular file or can get a catalogue of the files available. If a file contains confidential information (a company's accounts, for example, or the source code of a commercial program) it must be protected from unauthorised users. The simplest method involves locking the disk or tape away in a secure location, and some companies spend fortunes on this kind of security for their computer systems, recognising that the information contained is their most precious resource. Physical protection, however, is compromised by the need for people to use the system, and disks, tapes and security passes can go astray. Against these failures, therefore, the creator of a confidential file has to provide some form of built-in *file protection.*

A large multi-user system may provide the first level of protection by requiring users to *log-on* with an authorised user code and a user-defined password. The user code (sometimes called a PPN — programmer project number) determines the user's *system privileges,* including the level of access permitted. Each user has an individual file directory area on disk, and the proper combination of privilege and password is required to access other users' directories. System personnel, such as operators and programmers, often have common PPNs and passwords — like a hotel's passkeys — that can give access to the entire system. Because of the power this gives, and because the codes are often unimaginatively chosen, cracking this protection is a hacker's prime objective (see page 486).

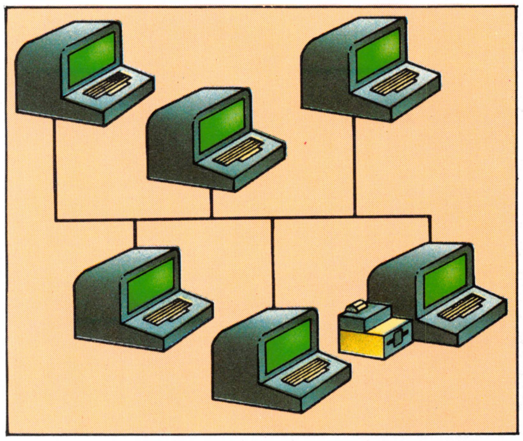
A more widespread problem than that of breaking into private data files is *software piracy* — the illegal copying of commercial program files. Software publishers have tried many forms of file protection, only to find most of them being broken as soon as they are introduced. The earliest file protection techniques included hiding data in normally unused spaces on a disk or tape. This method confuses the computer's operating system so that it can read the file only when the program is running, and cannot copy it. A more recent approach to this problem involves serialising disk-based software. The first time a program is run on a computer, a serial number associated with that particular computer is added to the file. From then on, that file — and any copies made from it — will work on that machine only.

FILE SERVER

In a network, one of the computers is sometimes set aside to handle file accesses for all the other terminals on the network. In such a system, all communications from and to the terminals, printers, disk drives and other peripherals are treated as separate files, which greatly simplifies the controlling computer's task. This computer is called a *file server,* and its only task is to control the movement of files between nodes on the network and the storage unit.

The file server is called by a node when a user needs a particular file (which could be a data or program file on disk, or a communication with a printer or other peripheral). If the file is open to that user and not being used by another node, the file server sends the data. The file is *massaged,* or altered, by the user and returned to the file server, where it is updated to incorporate any changes and then re-saved.

The Econet system is a low-cost and popular example of this kind of network, and normally it allows a BBC Micro as the file server. Users of the early versions of the system, however, found that more processor power was needed to run a network of any size. The Acorn System 3 was generally used as a stop-gap until the 6502 second processor for the BBC became available.



Joining Forces
The Acorn Econet joins up BBC Micros in a bus-type network. The bus may be a simple twisted-wire pair, and one micro is dedicated as the file server

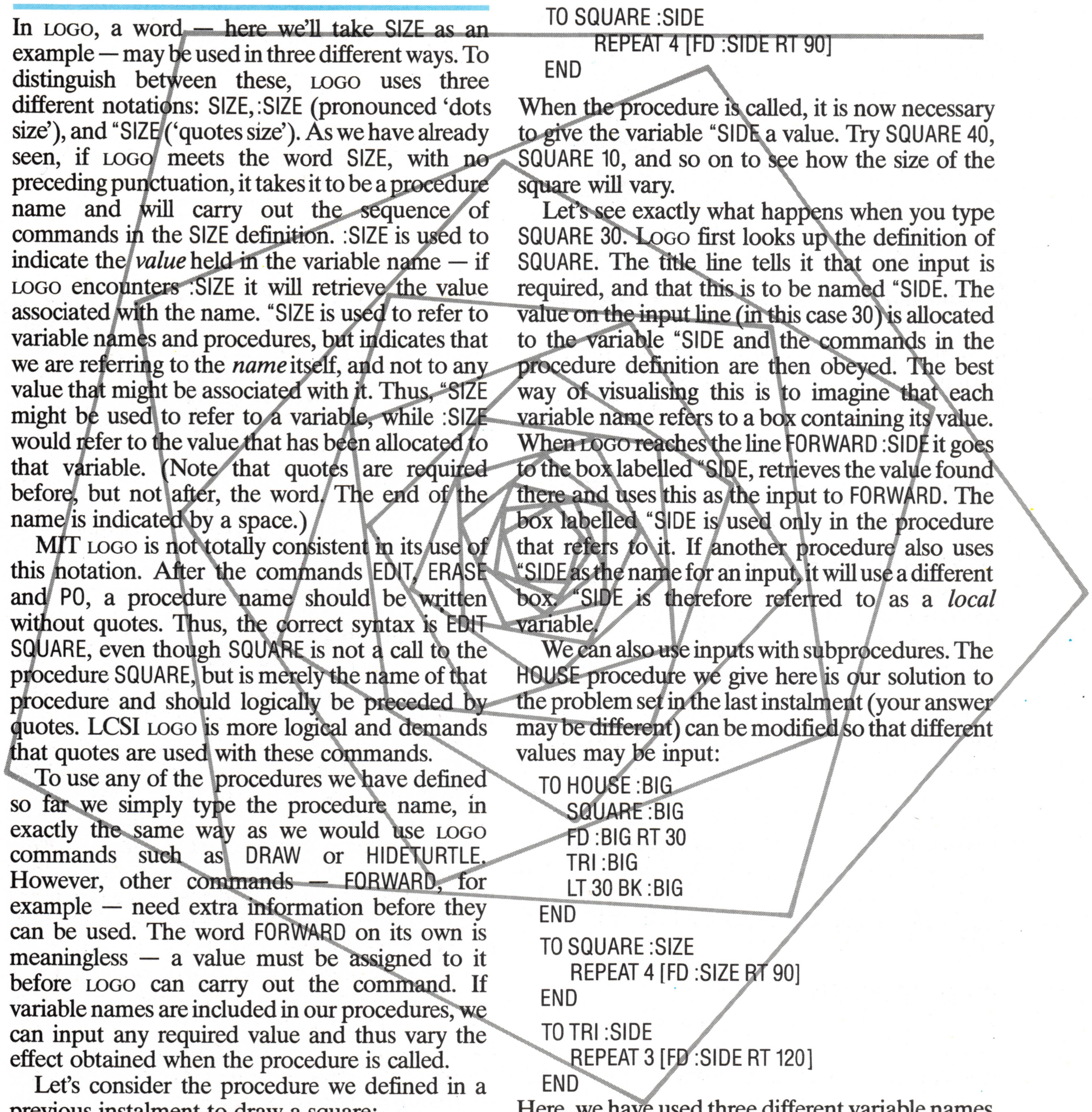
FILE TRANSFER

A file is usually created within a computer and then stored in some peripheral storage device. Copying the file between them — or to another node in a network, or to some distant system — are the commonest examples of *file transfer.* The problem in these vital communications is the recurring one of format compatibility. All communicating devices have their own requirements (or *protocol*) concerning the transmission rate, parity, and connect/disconnect signals. These protocols are often specific to the device, so file transfer is rarely a simple matter.

Different manufacturers and organisations have made various attempts at standardising these formats, but their solutions have usually added to the problems. The recent growth in national and international telephone data networks, however, is gradually imposing consistency. Within a network, the simplest answer is usually to make all transfers through the network controller (file server), which can accept a file from any node and pass it to any other node in the network in the appropriate format.

VARIATIONS ON A TURTLE

Our examination of the LOGO language has already shown us how procedures can be defined to carry out sequences of commands. Such procedures are more flexible if the user is able to input different values that will alter the effect achieved when the procedure is called.



In LOGO, a word — here we'll take SIZE as an example — may be used in three different ways. To distinguish between these, LOGO uses three different notations: SIZE, :SIZE (pronounced 'dots size'), and "SIZE ('quotes size'). As we have already seen, if LOGO meets the word SIZE, with no preceding punctuation, it takes it to be a procedure name and will carry out the sequence of commands in the SIZE definition. :SIZE is used to indicate the *value* held in the variable name — if LOGO encounters :SIZE it will retrieve the value associated with the name. "SIZE is used to refer to variable names and procedures, but indicates that we are referring to the *name* itself, and not to any value that might be associated with it. Thus, "SIZE might be used to refer to a variable, while :SIZE would refer to the value that has been allocated to that variable. (Note that quotes are required before, but not after, the word. The end of the name is indicated by a space.)

MIT LOGO is not totally consistent in its use of this notation. After the commands EDIT, ERASE and PO, a procedure name should be written without quotes. Thus, the correct syntax is EDIT SQUARE, even though SQUARE is not a call to the procedure SQUARE, but is merely the name of that procedure and should logically be preceded by quotes. LCSI LOGO is more logical and demands that quotes are used with these commands.

To use any of the procedures we have defined so far we simply type the procedure name, in exactly the same way as we would use LOGO commands such as DRAW or HIDETURTLE. However, other commands — FORWARD, for example — need extra information before they can be used. The word FORWARD on its own is meaningless — a value must be assigned to it before LOGO can carry out the command. If variable names are included in our procedures, we can input any required value and thus vary the effect obtained when the procedure is called.

Let's consider the procedure we defined in a previous instalment to draw a square:

```
TO SQUARE
  REPEAT 4 [FD 50 RT 90]
END
```

As it stands, this procedure will draw a square with sides of 50 units in length. However, it would be far more useful if the square could be drawn to any chosen size — to do this, we must input the desired value. To change SQUARE to accept an input, use the editor to replace the fixed value of 50 with the variable "SIDE and add :SIDE to the title line. Our procedure will now look like this:

```
TO SQUARE :SIDE
    REPEAT 4 [FD :SIDE RT 90]
END
```

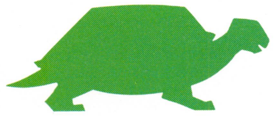
When the procedure is called, it is now necessary to give the variable "SIDE a value. Try SQUARE 40, SQUARE 10, and so on to see how the size of the square will vary.

Let's see exactly what happens when you type SQUARE 30. LOGO first looks up the definition of SQUARE. The title line tells it that one input is required, and that this is to be named "SIDE. The value on the input line (in this case 30) is allocated to the variable "SIDE and the commands in the procedure definition are then obeyed. The best way of visualising this is to imagine that each variable name refers to a box containing its value. When LOGO reaches the line FORWARD :SIDE it goes to the box labelled "SIDE, retrieves the value found there and uses this as the input to FORWARD. The box labelled "SIDE is used only in the procedure that refers to it. If another procedure also uses "SIDE as the name for an input, it will use a different box. "SIDE is therefore referred to as a *local* variable.

We can also use inputs with subprocedures. The HOUSE procedure we give here is our solution to the problem set in the last instalment (your answer may be different) can be modified so that different values may be input:

```
TO HOUSE :BIG
   SQUARE :BIG
   FD :BIG RT 30
   TRI :BIG
   LT 30 BK :BIG
END
TO SQUARE :SIZE
   REPEAT 4 [FD :SIZE RT 90]
END
TO TRI :SIDE
   REPEAT 3 [FD :SIDE RT 120]
END
```

Here, we have used three different variable names — "BIG, "SIZE and "SIDE. We could have used the same name for all three, as variables are local to the procedures in which they are used, but this could have been confusing.



To see how these procedures work, let's see what happens if we type HOUSE 30. LOGO reads the input line and assigns the value 30 to the variable "BIG in HOUSE. The first line of HOUSE is therefore now equivalent to SQUARE 30. The variable "SIZE in SQUARE is, in turn, assigned the value 30. SQUARE is now run, with FD :SIZE becoming FD 30. A similar procedure is followed when TRI is called.

Now try adapting the procedures for drawing the five-by five board so that BOARD takes the size of the square as an input.

Here's a procedure that draws polygons, with the number of sides given as an input:

```
TO POLY :SIDES
  REPEAT :SIDES [FD 50 RT 360 / :SIDES]
END
```

Using this procedure with one input, POLY 3 will draw a triangle, POLY 4 a square, and so on. However, in all the polygons drawn by this procedure, the sides will be 50 units in length. A more general procedure that draws polygons of any size requires two inputs — one for the number of sides, and one for the required length. To do this, all that is needed is to adapt the POLY procedure to replace the 50 with a variable name and add that name to the title line:

```
TO POLY :SIDES :SIZE
  REPEAT :SIDES [FD :SIZE RT 360/ :SIDES]
END
```

Now POLY 5 30 will draw a pentagon with sides of 30 units in length. You might like to try adapting your new version of the board-drawing procedure so that it will draw any square board (not just five-by-five). There will now be two inputs — the number of squares in each direction, and their size.

GLOBAL VARIABLES

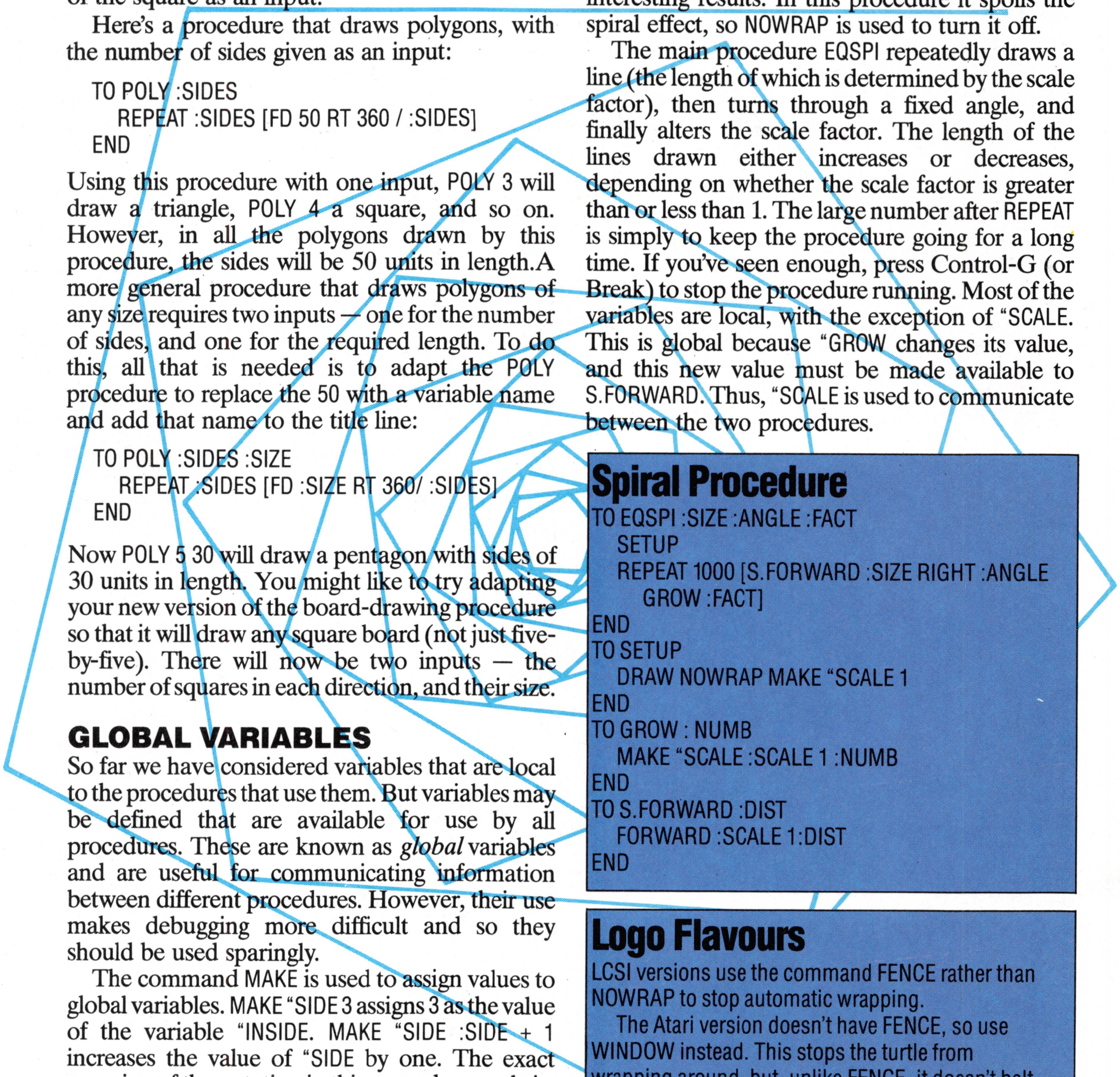
So far we have considered variables that are local to the procedures that use them. But variables may be defined that are available for use by all procedures. These are known as *global* variables and are useful for communicating information between different procedures. However, their use makes debugging more difficult and so they should be used sparingly.

The command MAKE is used to assign values to global variables. MAKE "SIDE 3 assigns 3 as the value of the variable "INSIDE. MAKE "SIDE :SIDE + 1 increases the value of "SIDE by one. The exact meaning of the notation in this second example is: find the value of the variable "SIDE, add one, then assign the result back to the variable named "SIDE. In each case, MAKE requires two inputs — the name of the variable, and the value to be assigned to that variable.

To sum up the programming features we have covered in this instalment of the LOGO course, we've designed some procedures for drawing spirals. The main procedure is named EQSPI. This requires three inputs: the initial length of the line to be drawn, the angle that must be turned at each 'corner' of the spiral, and a scale factor by which the initial length must be multiplied to produce the spiral effect. Different sets of inputs may be used to achieve different effects — we tried 70 283 0.95, 70 143 0.95, and 20 243 1.05. Try other sets of numbers and see what happens.

NOWRAP is a new command. This stops the turtle 'wrapping around' the screen — when the turtle reaches the screen boundary the procedure will stop with an 'out of bounds' error message. In many cases, the wrap-around effect can give interesting results. In this procedure it spoils the spiral effect, so NOWRAP is used to turn it off.

The main procedure EQSPI repeatedly draws a line (the length of which is determined by the scale factor), then turns through a fixed angle, and finally alters the scale factor. The length of the lines drawn either increases or decreases, depending on whether the scale factor is greater than or less than 1. The large number after REPEAT is simply to keep the procedure going for a long time. If you've seen enough, press Control-G (or Break) to stop the procedure running. Most of the variables are local, with the exception of "SCALE. This is global because "GROW changes its value, and this new value must be made available to S.FORWARD. Thus, "SCALE is used to communicate between the two procedures.



Spiral Procedure

```
TO EQSPI :SIZE :ANGLE :FACT
  SETUP
  REPEAT 1000 [S.FORWARD :SIZE RIGHT :ANGLE
    GROW :FACT]
END
TO SETUP
  DRAW NOWRAP MAKE "SCALE 1
END
TO GROW : NUMB
  MAKE "SCALE :SCALE 1 :NUMB
END
TO S.FORWARD :DIST
  FORWARD :SCALE 1:DIST
END
```

Logo Flavours

LCSI versions use the command FENCE rather than NOWRAP to stop automatic wrapping.

The Atari version doesn't have FENCE, so use WINDOW instead. This stops the turtle from wrapping around, but, unlike FENCE, it doesn't halt the procedure when the turtle reaches the screen boundary.

In the Spiral procedure on the Spectrum, replace DRAW in the SETUP subprocedure with CS.

Procedure Problems 2

1) Write a procedure to draw a circle of radius 50. Modify the procedure so that the radius is given as an input to the procedure.

2) Write a procedure that draws a 'target' consisting of five concentric circles.

Exercise Answers

1) Tangram Puzzles

The man sitting, the man bowing, and the cat shapes, all use the other side of the parallelogram piece from that used by the dog in last week's example. In order to turn a piece over in LOGO, you simply change all RIGHT turns to LEFT turns. So instead of:

```
TO PAR
   REPEAT 2 [FD 25 RT 45 FD 35 RT 135]
END
```

we will sometimes need its 'other side' given by:

```
TO PAR1
   REPEAT 2 [FD 25 LT 45 FD 35 LT 135]
END
```

All the other piece procedures are exactly as given in the last part of this course

Man Running

```
TO RUNNING
   MOVE1 TRI1 MOVE2 PAR MOVE3 TRI3 MOVE4
   TRI3 MOVE5 SQUARE MOVE6 TRI1 MOVE7 TRI2
   MOVE8
END
TO MOVE1
   LT 45
END
TO MOVE2
   PU FD 25 RT 135 FD 17.5 LT 45 PD
END
TO MOVE3
   PU FD 75 RT 90 PD
END
TO MOVE4
   PU RT 90 FD 25 RT 90 PD
END
TO MOVE5
   PU FD 50 RT 135 FD 50 LT 135 PD
END
TO MOVE6
   PU RT 135 FD 21 RT 135 FD 25 LT 90 FD 50
   LT 90 FD 25 RT 90 PD
END
TO MOVE7
   PU FD 25 RT 135 FD 71 RT 45 BK 35 PD
END
TO MOVE8
   PU FD 35 LT 90 FD 25 RT 45 FD 17.5 LT 45
   FD 25 RT 135 PD
END
```

Man Sitting

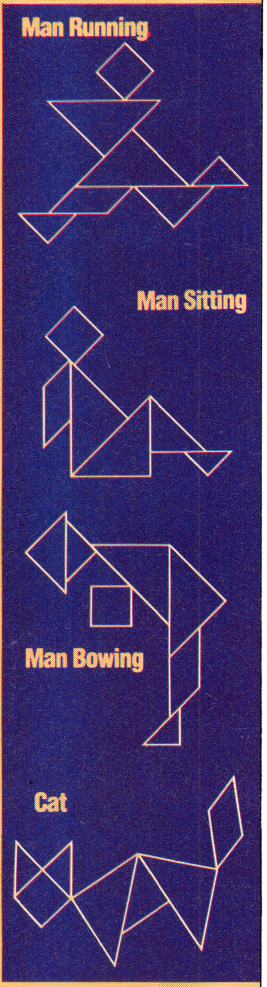
```
TO SITTING
   MOVE1 TRI1 MOVE2 TRI2 MOVE3 TRI3 MOVE4
   TRI1 PAR1 MOVE5 SQUARE MOVE6 TRI3 MOVE7
END
TO MOVE1
   LT45
END
TO MOVE2
   PU FD 25 LT 45 FD 17.5 RT 90 PD
END
TO MOVE3
   PU BK 15 LT 90 PD
END
TO MOVE4
   PU FD 50 RT 45 FD 25 RT 90 PD
END
TO MOVE5
   PU FD 25 LT 45 FD 35 LT 45 PD
END
TO MOVE 6
   PU BK 50 LT 90 PD
END
TO MOVE7
   PU BK 21 RT 135 BK 50 RT 90 FD 35 LT 90
   PD
END
```

Man Bowing

```
TO BOWING
   MOVE1 TRI1 MOVE2 PAR1 MOVE3 TRI3 MOVE4
   TRI3 MOVE5 TRI1 MOVE6 TRI2 MOVE7 SQUARE
   MOVE8
END
TO MOVE1
   LT 90
END
TO MOVE2
   PU FD 25 RT 135 FD 30 PD
END
TO MOVE3
   PU LT 45 FD 35 LT 135 BK 50 PD
END
TO MOVE4
   PU RT 90 FD 50 LT 135 PD
END
TO MOVE5
   PU RT 90 FD 50 LT 135 FD 5 RT 90 PD
END
TO MOVE6
   PU RT 90 FD 25 RT 45 FD 7.5 RT 45 BK 35
   PD
END
TO MOVE7
   PU FD 35 RT 135 FD 7.5 LT 45 FD 55 RT 45
   PD
END
TO MOVE8
   PU LT 45 FD 36 RT 45 FD 56 LT 135 FD 5
   LT 45 BK 25 PD
END
```

Cat

```
TO CAT
   MOVE1 TRI3 MOVE2 SQUARE MOVE3 TRI1 MOVE4
   TRI1 MOVE5 TRI3 MOVE6 PAR1 MOVE7 TRI2
   MOVE8
END
TO MOVE1
   PU FD 50 RT 90 PD
END
TO MOVE2
   RT 170
END
TO MOVE3
   PU RT 90 FD 25 LT 90 PD
END
TO MOVE4
   RT 180
END
TO MOVE5
   PU RT 90 FD 25 LT 80 FD 50 RT 45 FD 50
   RT 90 PD
END
TO MOVE6
   LT 155
END
TO MOVE7
   PU LT 160 FD 35 PD
END
TO MOVE8
   PU FD 35 LT 45 FD 21 RT 135 PD
END
```



These are our suggested answers to the Procedure Problems on page 565:

2)

```
TO HOUSE
   SQUARE FD 50 RT 30
   TRI LT 30 BK 50
END
TO SQUARE
   REPEAT 4 [FD 50 RT
   90]
END
TO TRI
   REPEAT 3 [FD 50 RT
   120]
END
```

3)

```
TO BOARD
   REPEAT 5 [LINE FD
   10] BK 50
END
TO LINE
   REPEAT 5 [SQUARE RT
   90 FD 10 LT 90] LT
   90 FD 50 RT 90
END
TO SQUARE
   REPEAT 4 [FD 10
   RT 90]
END
```

4)

```
TO SIX.STAR
   TRI MOVE TRI MOVE.
   BACK
END
TO TRI
   RT 30 REPEAT 3
   [FD 50 RT 120]
END
TO MOVE
   PU FD 29 RT 60 PD
END
TO MOVE.BACK
   PU LT 60 BK 29 PD
END
```

FINE TUNING

Our course on program design has so far concentrated on 'structured' programming methods. Using the techniques we have suggested will make your programs easier to design and debug, but will do nothing to make them run any faster. Here we consider ways to increase program execution speed.

Structured programming and good program layout are techniques that make programs easier to use, but do not improve program efficiency. To make programs run faster and use less memory space, it is often necessary to sacrifice clarity in a program's design. So we should bear in mind, when 'tuning up' a piece of code, that almost anything that is done to make it faster will invariably make it more difficult to read, understand and debug.

The inherent slowness of interpreted languages like BASIC means that there will be times when programs will run at an unacceptable pace and must be speeded up. The most efficient way of speeding up a BASIC program is to compile it. However, very few micros support a true BASIC compiler — there are disk- and cassette-based compilers on the market, but most of them support only integer BASIC, and may require special formatting of your program before compilation. Compiling is a slow process, especially during program development, and especially when the system is cassette-based. The compiler will occupy user memory, and the more comprehensive its facilities, the more RAM it will take from the user program area. In general on home micros, compiling is recommended only for fully tested and debugged programs.

File accesses slow programs down more than any other single cause. In a program that frequently reads from and writes to disk or tape (a database program, say) delays are inevitable. Access to a record in a random access file on a floppy disk takes an average of about a quarter of a second. Access to data in serial files takes longer (and varies with the length of the file) and tape accesses are considerably longer. If these delays are causing problems, it may be possible to reduce the number of accesses by reading in more data at once and storing it in RAM, and by 'saving up' updates to files until the end of the session. Interactive programs often cause problems because the user is left staring at a screen for several seconds. A partial solution here is to re-organise the program so that files are read and written while the user is busy doing something else (reading a screenful of instructions, for instance).

Another cause of slowness is real arithmetic. Real numbers are ones with decimal places (integers are whole numbers). Because of the decimal part, fetching a real number from memory and performing an arithmetical operation on it requires many more machine cycles than doing the same for an integer. In programs with a lot of arithmetic, it pays to replace all the variables involved with integer variables (e.g. SUM should be replaced by SUM%). Savings of around 20 per cent can be achieved for even moderately numerical programs and 'number-crunching' applications stand to gain by as much as 50 per cent.

Designing a faster algorithm is one of the best ways of speeding a program up. Some sources of algorithms have already been recommended in this course. Try these, and be on the lookout for those published in computer magazines. Otherwise, devising algorithms is a matter of creativity and insight. BASICS usually have a wealth of inbuilt functions (such as INSTR, SGN, LOG, and so on) that are very fast. This speed is a result of their being written in machine code and using the best algorithms available. It is often worth checking the manual again to see what inbuilt functions are offered before coding your own version. User-defined functions, implemented with the command DEF FN, also run quickly. This command is most useful in programs with repeated calculations or a repeated sequence of string manipulations, where it can replace a subroutine call, which is much slower.

Writing routines in machine code generally makes them run faster. This is because interpreted languages translate program lines into machine

Micro Compilers

BBC Micro Turbo Compiler Salamander, 17 Norfolk Road, Brighton BN1 3AA. 0273 771942	Cassette	£9.95
Commodore 64 DTL-BASIC Compiler Dataview Wordcraft Ltd, Radix House, East Street, Colchester CO1 2XB. 0206 86914	Cass/Disk	£14.95
Spectrum Softek FP Softek IS, Combined FP and IS, Softek International, 12/13 Henrietta St, London WC2. 01-240 1422	 Cass Cass Cass	 £19.95 £9.95 £24.95

code as they are encountered while the program is running (they do not do it particularly efficiently, either). Writing in machine code avoids this translation process. Unfortunately, writing Assembly language programs is much more difficult than writing BASIC, and the cost in time and effort may not be worth the eventual saving. However, some programs — those using animated graphics, for instance — would not work as intended if they were written in BASIC alone.

There are many other ways of making smaller savings in processing speed. Use a variable instead of an actual number (e.g. MAX rather than 267.5) for faster access to values, especially in loops. Use different letters to start variable names, and spread these initial letters evenly throughout the alphabet. Use multiple statement lines (if that is possible) and create a sizeable interval between line numbers (such as 10). With FOR . . . NEXT loops, if the interpreter permits, leave off the loop counter variable (for example, use NEXT rather than NEXT LOOP). Inside a loop, try to avoid calculating the same value over and over again. Instead, calculate it outside the loop and incorporate it as a variable.

SAVING SPACE

Integer arithmetic not only saves time, it also saves space. Where it may take four or five bytes to store a real number, it need take only two to store an integer. This represents a major saving, especially where large arrays are involved. Other improvements to the speed of a program will also save space: using inbuilt or user-defined functions saves code, as does writing in Assembly language and using multiple statement lines. Compiling tends to *increase* the size of smaller programs and only saves space for large ones.

Removing REM statements is an obvious space-saver, and using shorter strings of text for prompts also helps. Putting large blocks of text into files that are stored outside the program keeps them out of the way when they are not needed (instructions and 'help' files are the biggest burdens). Remove as many spaces as is legal within a line, and use shorter line numbers and shorter variable names. If an array needs to be dimensioned but its exact size is not known, don't just guess a convenient round number. Instead, leave it until the information needed is on hand and then dimension it with a variable, like this:

```
10 INPUT"How many instances are in this
   category?";INSTANCES%
20 DIM ARRAY%(INSTANCES%)
```

This is called 'dynamic dimensioning' and it is something that BASIC offers and most other languages don't — so make the most of it!

Another technique involves increasing BASIC's memory allocation in RAM. This can be done by using commands like HIMEM. What these commands usually do is to change the area in RAM that is available to BASIC programs and variables. The normal use for this is to store machine code programs in a safe place where they won't be overwritten, but the same command can be used to access extra space from that normally reserved for the screen memory. If it does not matter what is appearing on the screen, then this is a good way to get an extra kilobyte of RAM. If it is not possible to change HIMEM, the screen memory can still often be used by PEEKing and POKEing directly to the memory locations reserved for it.

If all else fails and the program simply will not fit in the space available, many BASICS have a CHAIN command that allows one program to pass control to another. Some BASICS allow use of the COMMON command; this passes particular variables and their current values to the next program. CHAIN on home micros (if it exists at all) is usually a very simple command that enables all or none of the variables from the first program to be passed to the second.

If programs are written in a structured way, the individual subroutines should be capable of being written and tested independently. Their execution can also be individually timed. Write a simple timer like this one:

```
100 REM Use this first section to set any variables
105 REM that the routine will need (don't forget
110 REM to dimension arrays and fill them with
115 REM realistic data too if the routine uses any).
120 REM This program is in BBC BASIC and TIME
125 REM is a pseudo-variable that holds a value in
130 REM hundredths of a second, generated by the
135 REM system clock
200 START=TIME
210 GOSUB 2000:REM The routine being timed is
    called here.
220 FINISH=TIME
230 PRINT "Execution took"; (FINISH-START)/100;
    "seconds."
240 END
```

With this routine it is possible to experiment with different algorithms and other ways of increasing speed.

How To Be Quick

- Weigh carefully the demands of good style against the need for speedy, sometimes incomprehensible, code.
- Compile when you can; define functions and procedures if you can't.
- Avoid file accesses.
- Avoid absolute real numbers. Initialise variables and use integer arithmetic, if your micro allows it.
- Design your algorithms carefully and learn from the example of others.
- Consider the advantages and disadvantages of machine code. While it may be fast, it takes longer to write and to debug.
- Condense your code, and remove your REMs once you have a working version.

CHANGE OF ADDRESS

At this stage in the course, we take a detailed look at how the two index registers, X and Y, are used in indexed addressing. We illustrate the value of indexing by reference to several example programs.

The original concept of the stored program computer was that by storing the program in the same place (and in the same form) as the data on which it was to operate, the program could modify itself as it was running. The major use of this feature was not to modify the actual instructions themselves but to modify the addresses where the instructions got their data. Imagine the problem of having to access a table of several thousand numbers, and having to give separate instructions for each because each instruction could refer only to the one unchangeable address.

This problem was greatly alleviated with the introduction of the concept of *address modification*. In this way, the same instruction could be repeated any number of times and be made to refer to different addresses where data was stored by using the changing value in a register to alter the address. We use this sort of concept all the time in BASIC programs. For example:

```
FOR I = 1 TO N
PRINT TABLE(I)
NEXT I
```

In this case, the same PRINT instruction refers to different data each time it is used by modifying the basic data item (TABLE) using an indexed value (I), which is changed each time it is used.

The fundamental principle of *indexed addressing* is that the contents of the index register are added to the base address given in the instruction to produce the *effective address* — the memory location that is actually accessed. If this instruction occurs within a loop, then the adjustment to the index register (usually an increment or decrement) can be performed within the loop as well, and thus we can easily access a whole table of values.

The 6809 not only has two registers for this purpose, the X and Y registers, but also two further registers — S and U. In special circumstances it is possible to use the program counter as well. To make the whole subject of indexed addressing even more complex, there are a variety of different modes of indexing. However, these do cover nearly all programming requirements. We shall be using indexing, in one form or another, in all our programming from this point, so there will be plenty of opportunity for you to familiarise yourself with the variety of ways in which it is used.

Indexed addressing is indicated by adding ,X to the operand field — if the register used is the X register, of course. So the general form of an indexed instruction is:

```
Opcode  Offset,Index Register
  LDA   TABLE1,X
  STA   TABLE2,Y
```

In many situations the offset is zero, in which case it can be omitted. For example:

```
Opcode  ,Index Register
  LDA   ,X
  STA   ,Y
```

Let's see how this works in practice. Suppose we have a table of 64 eight-bit values stored at $3000 and we want to access the bytes in sequence. We can define the base address and reserve space for the table using the directives:

```
              ORG        $3000
TABLE         RMB        64
```

These instructions set the program counter to $3000, define TABLE as beginning at $3000, and reserve the next 64 bytes. We now access the bytes using the next piece of code: the new ORG directive means that our code will be stored in a different part of memory from our data. When you start using indexed addressing this is a sensible precaution against a loop getting out of control and causing your program to overwrite itself.

```
              ORG        $1000
COUNT         FCB        0
              LDX        #0
LOOP          LDA        TABLE,X
```

We now alter the value in the X register:

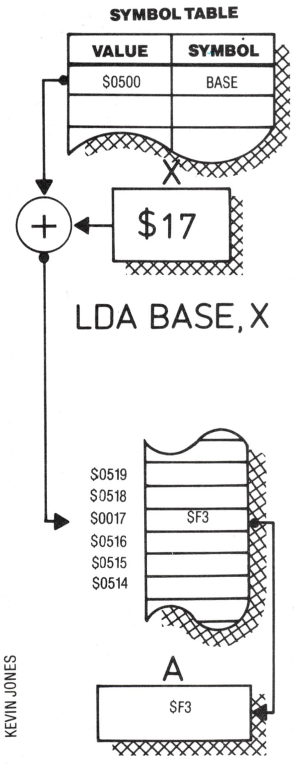
```
              TFR        X,D
              ADDD       #1
              TFR        D,X
```

This is an awkward way of incrementing X, although it can be useful when we are incrementing or decrementing by numbers greater than two. We will look at the alternatives to the method used here later in the course. The last fragment of code increments the count and checks to make sure that it is not 64 (in which case the program is finished and there is no need to loop again):

```
              INC        COUNT
              LDB        COUNT
              CMPB       #64
              BLT        LOOP
```

There are a number of ways of improving the efficiency of this code. One of the most useful

Index Links
The symbol BASE — initialised as $0500 — is the address of the first byte in a table of values. The indexed addressing mode instruction LDA BASE,X takes the value of BASE, and adds to it the contents of the X register, to produce the actual address of the byte whose contents are loaded into the accumulator. If this instruction is inside a loop of which X is the counter, then the entire table can be accesssed, byte by byte, in sequence. Since X is a 16-bit register, the loop could range over the entire memory space ($0000 — $FFFF in an eight-bit system such as the 6809)



KEVIN JONES

ways is to use the *auto-increment* mode. The instruction:

```
            LDA          TABLE,X+
```

will cause the value in X to be automatically incremented after it has been used. If we have a table of 16-bit values then we use:

```
            LDA          TABLE,X++
```

which causes the X register to be incremented by two. Our original program loop is now considerably streamlined:

```
LOOP        LDA          TABLE,X+
            INC          COUNT
            LDB          COUNT
            CMPB         #64
            BLT          LOOP
```

Another useful alternative to the original method we outlined is to step through the table of values in the reverse order, perhaps using the auto-decrement mode. This has the advantage that the final value in the X register is zero, and as auto-decrement of an index register automatically sets the flags in the condition code register, we can test directly for the end of the loop without having to use a CMP instruction. The same effect can be obtained by loading the index register with a negative value and incrementing this until zero is reached. Every time the auto-increment instruction is obeyed, it sets the condition code register flags to show the results of the increment. If a zero result occurs, for example, the zero flag is set; if a carry occurs, then the carry flag is set, and so on.

We should remember, however, our general rule that it is always best to make tests on the accumulators only. Also, since most programs are likely to have some processing between the increment/decrement instruction and the test instruction it is unlikely that the condition code register will remain unchanged between the action and the test.

If we decide not to step backwards through the table, it is still a good idea to make the count go backwards so that we can end the loop at zero. A point to watch with the auto-decrement instruction mode is that the decrementing is performed *before* the address calculation, whereas in auto-increment the register is incremented *after* the address has been calculated. Thus, if X contains 7 and TABLE begins at $1000, then the instruction LDA TABLE,X+ will load the accumulator from address $1007 and then increment X from 7 to 8. LDA TABLE,-X (note the minus sign comes before the register name), on the other hand, will decrement X from 7 to 6 and then load the accumulator with the value from address $1006.

Stepping through the table backwards, and keeping the count in the B register for convenience, our loop will now be:

```
       LDX    #64
       LDB    #64
LOOP   LDA    TABLE,-X
       DECB
       BGE    LOOP
```

The first of our two example programs shows a straightforward loop through a table of eight-bit values, in which we count the number of negative values. The count in accumulator B is also used as the offset from a fixed value in X. The second program shows both the index registers being used together, with a zero offset. It makes a copy of a character string from one location (possibly an input buffer) to another location where it will be stored. The string is of unknown length (although it will be less than 255 bytes) and will terminate with a Return character. When it is stored, the Return character will be deleted, and a byte indicating the string's length will be put at the beginning.

Negative Values Counter

● A table of eight-bit values is stored at location $3001. The number of values in the table is stored at $3000 and is assumed to be less than 256. This program counts the number of negative values in the table

```
TABLE   EQU   $3000     TABLE contains the number of
        ORG   $1000     values
NEGS    FCB   0         NEGS is somewhere to keep
        LDB   TABLE     the count of negative values
        LDX   #TABLE    The address of the table goes
                        into X
LOOP    LDA   B,X       Get the last value from the
                        table
        BGE   ENDLP     GOTO ENDLP if value is
                        positive
        INC   NEGS      ELSE count it if the value is
                        negative
ENDLP   DECB            Step down through table
        BGT   LOOP      Any more values? If so, GOTO
                        LOOP
        SWI             ELSE return to operating
        END             system
```

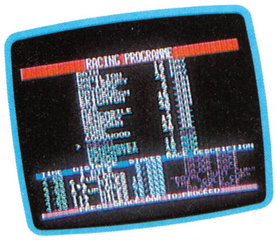
Character String Copier

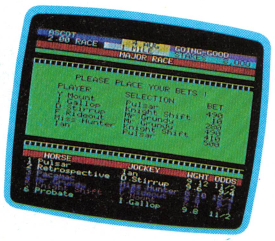
● This program copies a string from the input buffer at $3000 to the next free string space, the address of which is given at $0010

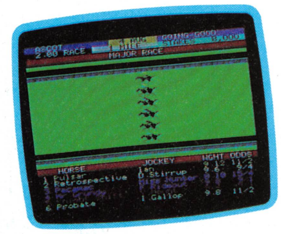
```
CR      EQU   13        ASCII value of carriage return
                        character
STRING1 EQU   $3000     Address of the input buffer
STRING2 EQU   $0010     STRING2 holds the address of
        ORG   $1000     the free space
        LDX   #STRING1  Address of source string (i.e.
                        $3000 at first) loaded into X
        LDY   STRING2   Address of destination string
                        (i.e. address stored at
                        $10,$11) into Y
        TFR   Y,U       Transfer the address of the
                        length byte into U
        LDA   #0        Store zero at first byte of
        STA   ,Y+       destination string
LOOP    LDA   ,X+       Get next character from input
                        buffer
        CMPA  #CR       Is it a Return character?
        BEQ   FINISH    Stop if it is
        STA   ,Y+       ELSE copy it (and)
        INC   ,U        Add one to the length
        BRA   LOOP      Next character
FINISH  SWI             Return to operating system
END
```

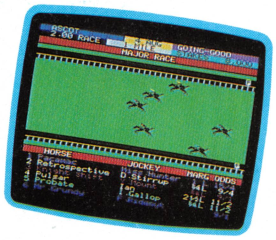
ODDS-ON FAVOURITE

The Sport of Kings — or just a mug's game? Opinions may vary as to the merits of horse racing, but Salamander Software's Classic Racing for the Oric-1 and Atmos home computers is a clear winner in the flat race simulation stakes.









Winning Stages
At any stage of the game, the players may consult the racing programme (top picture). This gives details of the race venues, and also indicates the prize money available for specific races. After the runners have been chosen for a particular race, players have the opportunity to bet on the horse of their choice (second picture). Bets must be between £5 and £500. The horses are then lined up to wait for the starter's gun, and the players must sit back and wait while the race is run. Our final picture shows the horses slowing down after passing the winning post, with our photographer's selection finishing a poor fourth . . .

Classic Racing gives you the opportunity to play the part of a race horse trainer over a season of meetings. It is a game for one to six players, but if there are fewer than six people playing, then the computer makes up the numbers so that each race will have six runners. The game allows you to choose the length of the season: a full season covers 16 race meetings, each consisting of six races. Those with less stamina may select a shorter season. The player's objective is to make as much money as possible. This may be achieved in two ways: you can collect prize money by training one of the first three horses in a race, or you may bet on the result. You *must* enter a runner for each race, but there is nothing to stop you betting on one of your opponents' runners if you feel this gives you a better chance to win money.

Your stable comprises 16 horses, and at the beginning of the season you have no idea of their merits. As early season races are for small prizes, this is the ideal time to experiment by trying your runners over different distances and in different ground conditions. This is simply a matter of trial and error — you must observe how a particular horse performs under given conditions and plan your strategy accordingly. This does entail copious note-taking — each time a horse runs you need to jot down the distance, the weight carried, the 'going' (ground condition), and the result. It's a pity that Salamander has failed to include a routine to print such details out automatically, as this would save a lot of effort.

Once you have chosen all six of your runners for the first meeting, you will be given the names of your opponents' horses and told the weight each will be carrying. The computer then allocates odds against each horse winning. At the start of the season this appears to be done in an arbitrary manner but, as the season wears on, horses with proven track records will start at shorter odds. Betting is compulsory — stakes must be between £5 and £500 — and the odds offered can be very generous. Because a winning (or placed) horse will collect prize money, it is often profitable to bet on an opponent's runner, thus giving you two chances of making a profit.

It is also possible to engineer betting 'coups', by entering a horse in races for which it is obviously unsuitable — for example, a horse that performs well over five furlongs in heavy going may be entered for two successive $1\frac{1}{2}$ mile races on firm ground. It will almost certainly lose ignominiously, and then may be entered in a more suitable race at good odds. However, once you have ascertained the ideal distance and going for a particular horse, you must resist the temptation to keep running it in race after race — as in the real-life racing world, horses need to be rested every so often if they are to perform at their best.

When all the bets have been placed, the action switches to the race itself. The horses amble into position, the starter calls them to order, then the Oric sound facilities produce a fair approximation of hoofbeats on turf as the runners head for the winning post. The race sequence is beautifully done: horses jostle for position in a realistic way and the runners are just as prone to erratic behaviour as their real-life counterparts are. It is infuriating to have to sit back and watch as your selection slows to a walk a hundred yards from the finish while the odds-on favourite glides past!

At the end of the race, winning bets are paid and the process is repeated for the rest of the card until the end of the meeting. Each meeting has a track with different ground conditions and race distances. If you eventually decide that one of your horses is not up to standard it may be dropped from your roster by simply failing to race it at three successive meetings. This gives you one less factor to worry about, but it costs you a £1,000 penalty at each remaining meeting.

Towards the end of the season, the races become harder to win, as all the players then have a much better idea of their horses' capabilities and are less likely to enter runners in races they have no chance of winning. The rewards are correspondingly greater — the first three home in the Derby, which is run during the last meeting of the season, share £90,000 in prize money.

Classic Racing is the most impressive piece of software yet for the Oric and Atmos. The race sequences are compelling viewing, and the strategy involved in planning your season's campaign makes this a game that will hold your interest over repeated playing. It's possible to win more than £250,000 over the full season — the only problem is collecting your winnings!

Classic Racing: For 48K Oric-1/Atmos, £6.95
Publishers: Salamander Software, 17 Norfolk Road, Brighton BN1 3AA
Author: Paul Neal
Joysticks: Not required
Format: Cassette

DATABASE

Here, courtesy of Zilog Inc., we reproduce the first part of the Zilog Z80 programmers' reference card.

MAIN REG SET		ALTERNATE REG SET		
ACCUMULATOR A	FLAGS F	ACCUMULATOR A'	FLAGS F'	
B	C	B'	C'	GENERAL PURPOSE REGISTERS
D	E	D'	E'	
H	L	H'	L'	

INTERRUPT VECTOR I	MEMORY REFRESH R	SPECIAL PURPOSE REGISTERS
INDEX REGISTER IX		
INDEX REGISTER IY		
STACK POINTER SP		
PROGRAM COUNTER PC		

Instruction	D7 S	Z		H		P/V	N	D0 C	Comments
ADD A, s, ADC A, s	↕	↕	X	↕	X	V	0	↕	8-bit add or add with carry
SUB s, SBC A, s, CP s, NEG	↕	↕	X	↕	X	V	1	↕	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	↕	↕	X	1	X	P	0	0	Logical operations
OR s, XOR s	↕	↕	X	0	X	P	0	0	
INC s	↕	↕	X	↕	X	V	0	•	8-bit increment
DEC s	↕	↕	X	↕	X	V	1	•	8-bit decrement
ADD DD, ss	•	•	X	X	X	•	0	↕	16-bit add
ADC HL, ss	↕	↕	X	X	X	V	0	↕	16-bit add with carry
SBC HL, ss	↕	↕	X	X	X	V	1	↕	16-bit subtract with carry
RLA, RLCA, RRA, RRCA	•	•	X	0	X	•	0	↕	Rotate accumulator
RL m, RLC m, RR m, RRC m, SLA m, SRA m, SRL m	↕	↕	X	0	X	P	0	↕	Rotate and shift locations
RLD, RRD	↕	↕	X	0	X	P	0	•	Rotate digit left and right
DAA	↕	↕	X	↕	X	P	•	↕	Decimal adjust accumulator
CPL	•	•	X	1	X	•	1	•	Complement accumulator
SCF	•	•	X	0	X	•	0	1	Set carry
CCF	•	•	X	X	X	•	0	↕	Complement carry
IN r (C)	↕	↕	X	0	X	P	0	•	Input register indirect
INI, IND, OUTI, OUTD	X	↕	X	X	X	X	1	•	Block input and output. Z = 0 if B ≠ 0 otherwise Z = 0
INIR, INDR, OTIR, OTDR	X	1	X	X	X	X	1	•	
LDI, LDD	X	X	X	0	X	↕	0	•	Block transfer instructions. P/V = 1 if BC ≠ 0, otherwise P/V = 0
LDIR, LDDR	X	X	X	0	X	0	0	•	
CPI, CPIR, CPD, CPDR	X	↕	X	X	X	↕	1	•	Block search instructions. Z = 1 if A = (HL), otherwise Z = 0. P/V = 1 if BC ≠ 0, otherwise P/V = 0.
LD A, I, LD A, R	↕	↕	X	0	X	IFF	0	•	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag.
BIT b, s	X	↕	X	1	X	X	0	•	The state of bit b of location s is copied into the Z flag

Symbol	Operation
S	Sign flag. S = 1 if the MSB of the result is 1.
Z	Zero flag. Z = 1 if the result of the operation is 0.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V = 1 if the result of the operation is even, P/V = 0 if result is odd. If P/V holds overflow, P/V = 1 if the result of the operation produced an overflow.
H	Half-carry flag. H = 1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator.
N	Add/Subtract flag. N = 1 if the previous operation was a subtract.
H & N	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
C	Carry/Link flag. C = 1 if the operation produced a carry from the MSB of the operand or result.
↕	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care."
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range < 0, 255 >.
nn	16-bit value in range < 0, 65535 >.

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
LD r, r'	r ← r'	•	•	X	•	X	•	•	•	01 r r'		1	1	4	r, r' Reg.
LD r, n	r ← n	•	•	X	•	X	•	•	•	00 r 110		2	2	7	000 B
										← n →					001 C
LD r, (HL)	r ← (HL)	•	•	X	•	X	•	•	•	01 r 110		1	2	7	010 D
LD r, (IX + d)	r ← (IX + d)	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	011 E
										01 r 101					100 H
										← d →					101 L
LD r, (IY + d)	r ← (IY + d)	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	111 A
										01 r 110					
										← d →					
LD (HL), r	(HL) ← r	•	•	X	•	X	•	•	•	01 110 r		1	2	7	
LD (IX + d), r	(IX + d) ← r	•	•	X	•	X	•	•	•	11 011 101	DD	3	5	19	
										01 110 r					
										← d →					
LD (IY + d), r	(IY + d) ← r	•	•	X	•	X	•	•	•	11 111 101	FD	3	5	19	
										01 110 r					
										← d →					
LD (HL), n	(HL) ← n	•	•	X	•	X	•	•	•	00 110 110	36	2	3	10	
										← n →					
LD (IX + d), n	(IX + d) ← n	•	•	X	•	X	•	•	•	11 011 101	DD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD (IY + d), n	(IY + d) ← n	•	•	X	•	X	•	•	•	11 111 101	FD	4	5	19	
										00 110 110	36				
										← d →					
										← n →					
LD A, (BC)	A ← (BC)	•	•	X	•	X	•	•	•	00 001 010	0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	X	•	X	•	•	•	00 011 010	1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	X	•	X	•	•	•	00 111 010	3A	3	4	13	
										← n →					
										← n →					
LD (BC), A	(BC) ← A	•	•	X	•	X	•	•	•	00 000 010	02	1	2	7	
LD (DE), A	(DE) ← A	•	•	X	•	X	•	•	•	00 010 010	12	1	2	7	
LD (nn), A	(nn) ← A	•	•	X	•	X	•	•	•	00 110 010	32	3	4	13	
										← n →					
										← n →					
LD A, I	A ← I	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
										01 010 111	57				
LD A, R	A ← R	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
										01 011 111	5F				
LD I, A	I ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
										01 000 111	47				
LD R, A	R ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
										01 001 111	4F				

NOTES r, r' means any of the registers A, B, C, D, E, H, L
IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

IMAGES PRODUCED ON THE ARTRON 2000 STUDIO COMPUTER; COURTESY OF GRAPHIC PRODUCTS, LONDON (EXCLUSIVE DISTRIBUTOR)